CMPM 120

Version Control, Debugging, & Endless Runners

Schedule Overview*

6/22	Introduction
6/24	Programming Our First Phaser Game
6/29	Version Control & Physics
7/1	Input and Movement
7/6	Physics and Debugging
7/8	State Machines & Cameras
7/13	JSON, Tilemaps, Map Editors
7/15	Tweens & Particles
7/20	Special Topics
7/22	Final Presentations

*This will inevitably change a bit

Schedule Overview*

6/22	Introduction		
6/24	Programming Our First Phaser Game	Rocket Patrol Tutorial Due	6/26
6/29	Version Control & Physics	Rocket Patrol Mods Due	6/29
7/1	Input and Movement		
7/6	Physics and Debugging	Endless Runner Due	7/6
7/8	State Machines & Cameras		
7/13	JSON, Tilemaps, Map Editors	Final Game: First Build	7/13
7/15	Tweens & Particles		
7/20	Special Topics		
7/22	Final Presentations	Final Game Due	7/22

*This will inevitably change a bit

Rocket Patrol

How are people doing?

Q:

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

Common issues

- In VSCode Git never finishes, cloud icon just keeps spinning
  - Most common cause: not being logged into Git
  - Open a terminal and type: `git push -u origin main`
    - It should ask you to open a browser and log in
  - Other git-related solutions: https://canvas.ucsc.edu/courses/44176/discussion_topics/281966
-

Getting Help

Post in the Discord

Post in Canvas Discussions

Answer questions other people ask!

Check the Phaser documentation:
https://photonstorm.github.io/phaser3-docs/index.html

Look at the Phaser examples and tutorials: https://phaser.io/learn

The Week Ahead

- Tuesday, June 29, 9am
  - Eloquent JavaScript: Functions, Data Structures, Objects, & Classes
  - Rocket Patrol Mods [~10–15 hours]
- Thursday, July 1
  - Understanding the JavaScript keyword "this"
- Friday, July 2
  - Character Movement
  - Game Loops in JavaScript
  - **Rocket Patrol Tutorial Feedback**
- Tue Jul 6, 2021
  - Endless Runner [~20-30 hours]

- Remember that the deadlines for the Readings are suggested times when they would be helpful to know, not hard requirements!

Submitting Projects

- At the deadline **you should submit whatever you have finished so far**.
- If it isn't done, feel free to leave a note that it isn't finished.
- You can continue to work on the project, and get it regraded when you resubmit it.
- If the Canvas is locked, email me.
- Work submitted after the deadline will get less feedback.
- There's a hard cutoff at July 23rd because I do have to turn grades in eventually.

Version Control

Version Control

Version control tracks and manages your software (and other) project history, including collaboration with team members.

Life Before Version Control

game-backup.js

game.js

gameNEW.js

gameNEWv2.js

gameNEWv2.1.js

gameRealNew.js

game-fixes-v2.js

game-final.js

game-final-fantasy-xv.js

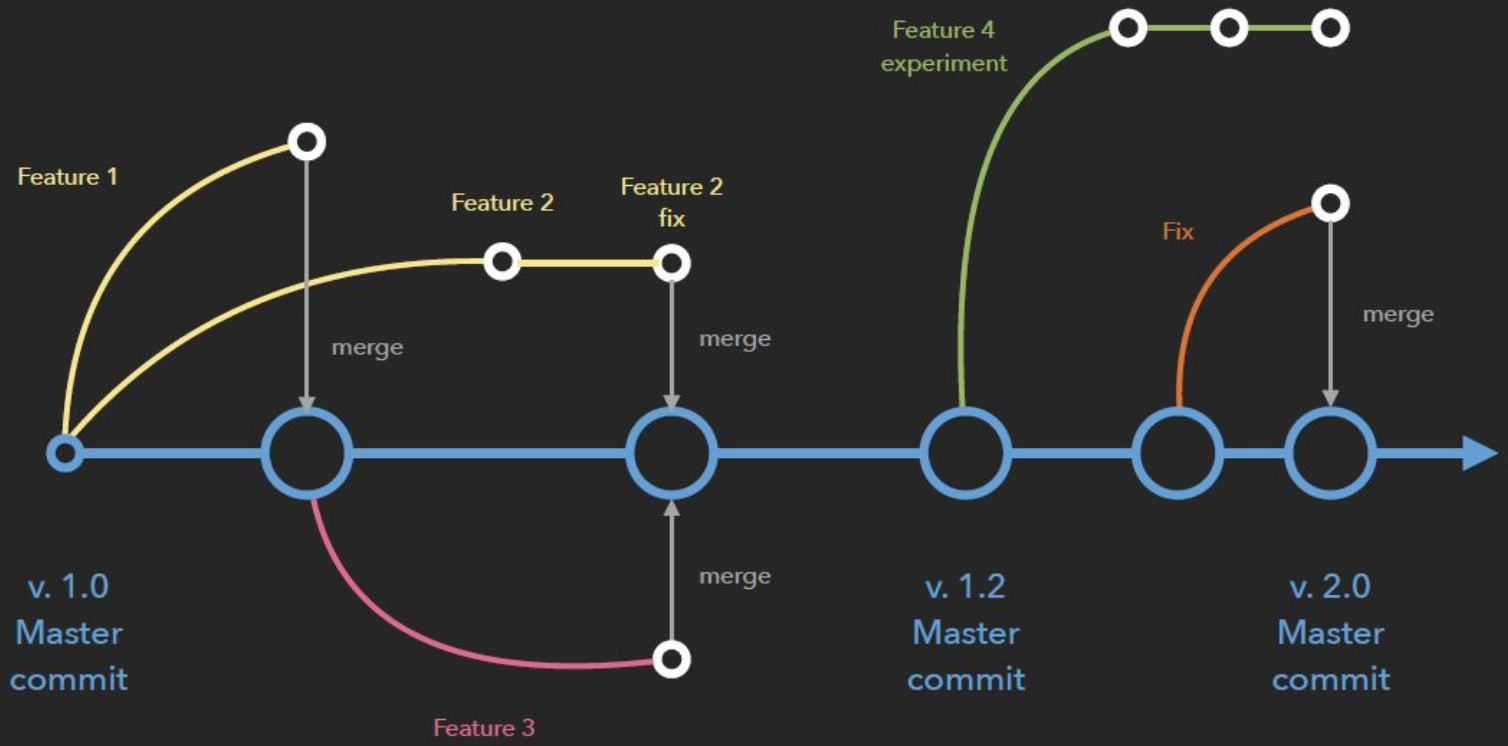
game-final-dev.js

game-release.js

game-release-actual.js

game-FINAL-deathAwaits.js

game-LAST-FINAL-Dream-Drop-Distance.js


Feature 4
experiment
Feature 1
Feature 2
Feature 2
fix
Fix
merge
merge
merge
merge
v. 1.0
Master
commit
v. 1.2
Master
commit
v. 2.0
Master
commit
Feature 3

Terms

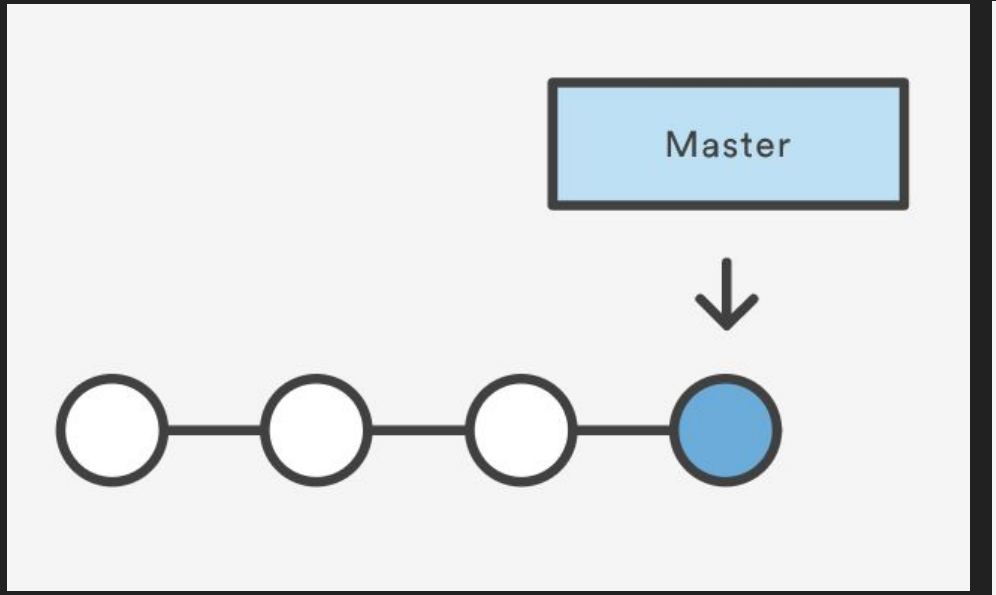
repository: all the files and folders associated with a project, along with each file's revision history. (A "repo")

commit: add new content, a "snapshot in time"

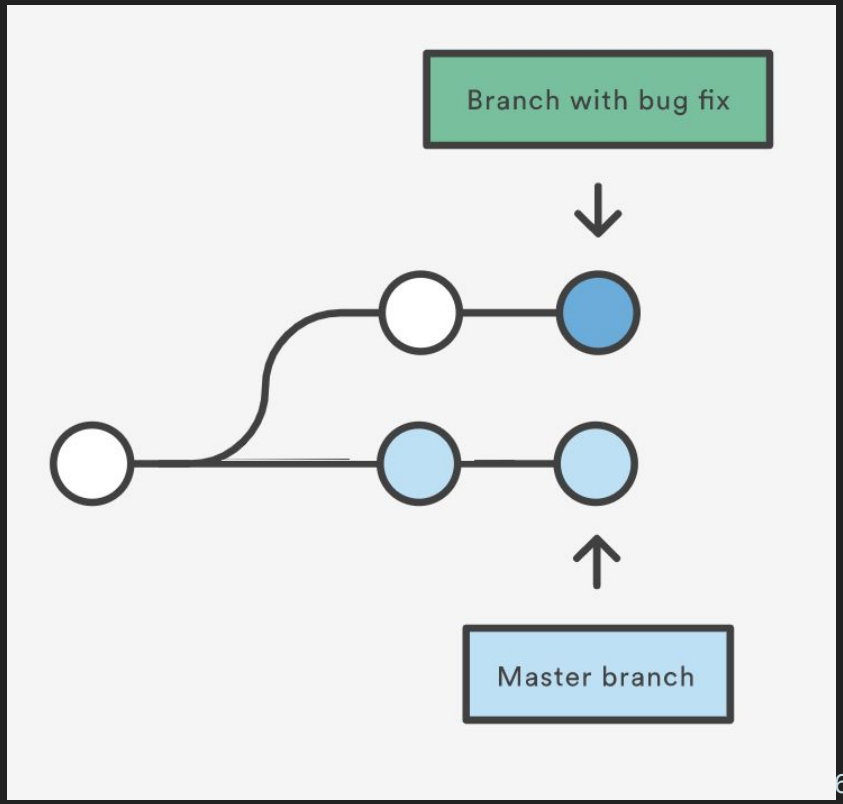
branch: new segment of development history

merge: bring changes from one branch into another

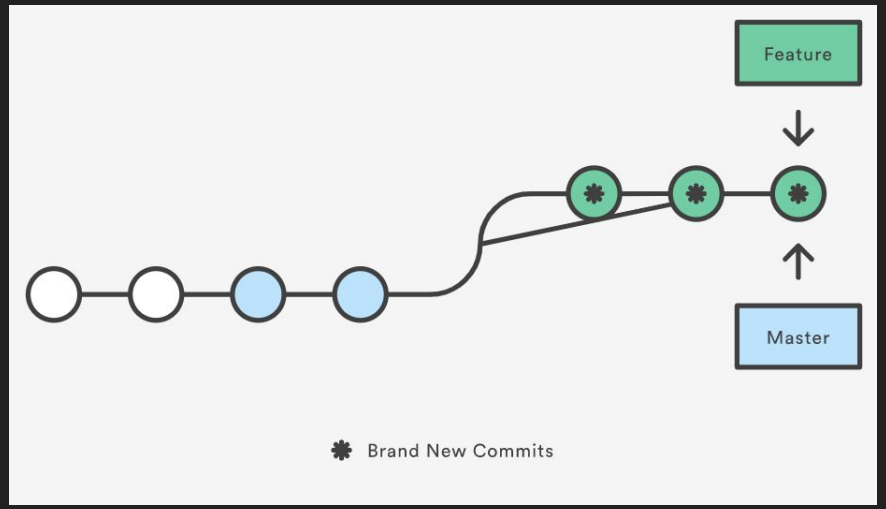
How do we fix bugs without interrupting mainline work?

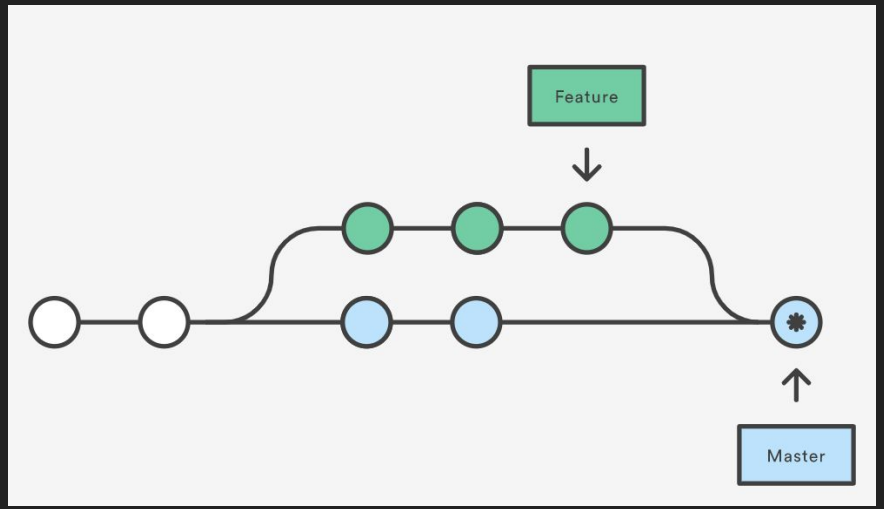


If we all take turns and make one change at a time, the development history looks nice and clean. But...



There's more than one way to do it.

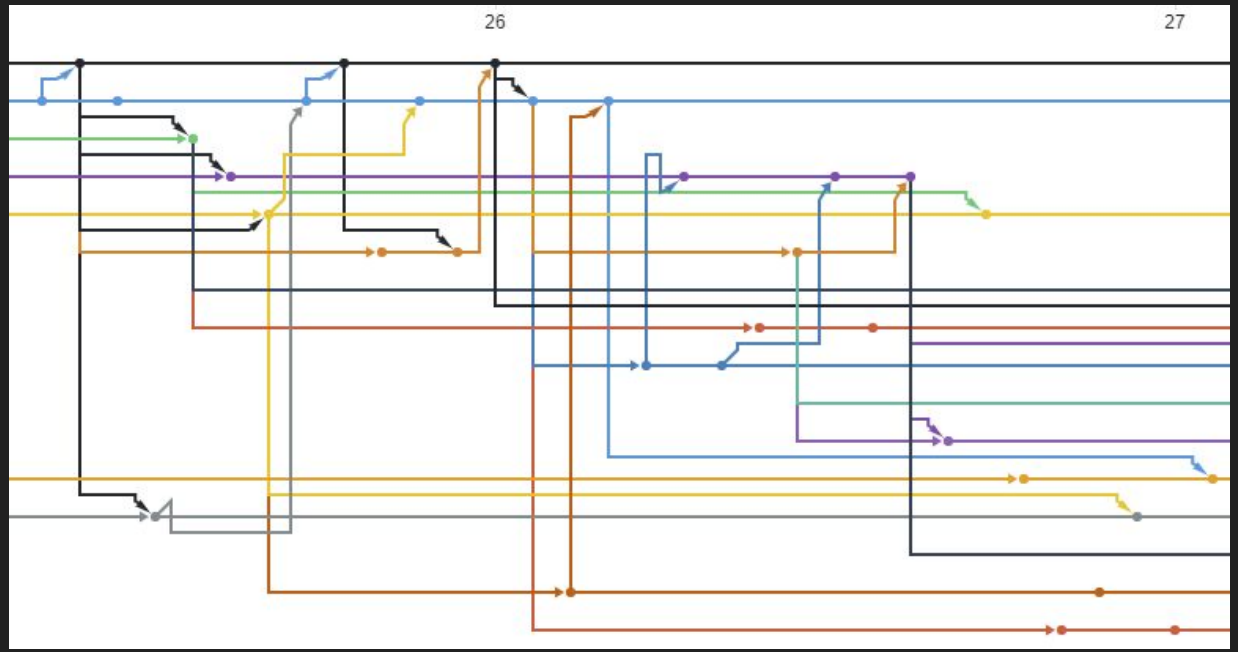




Rebasing, merging, oh my!

Branching

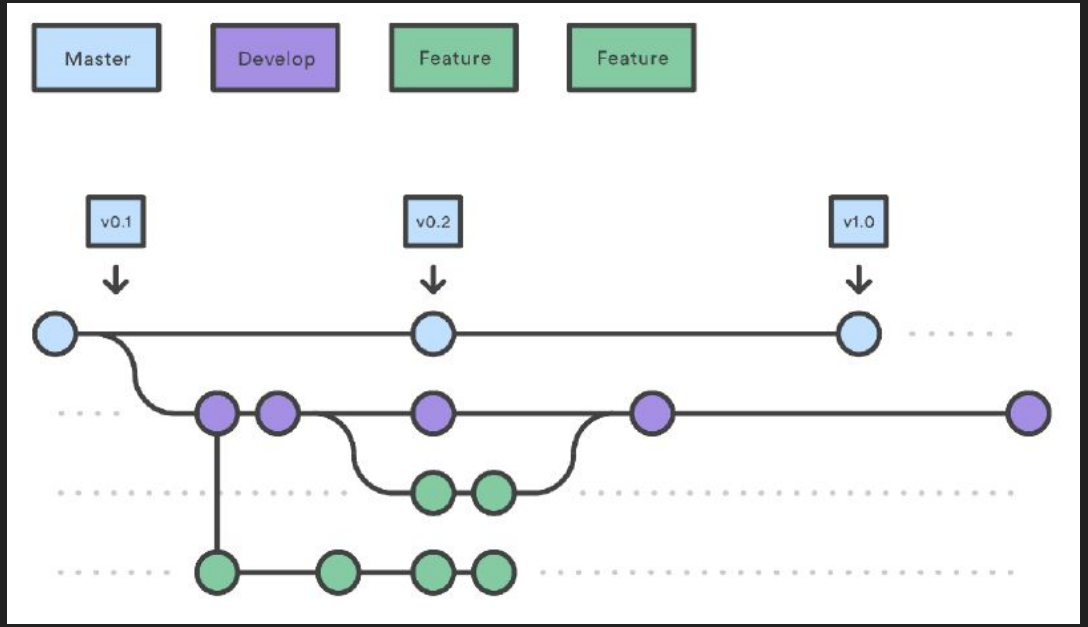
Git organizes repositories through **branching** and **merging**



https://github.com/ikarth/game-boy-rom-generator/network

GitHub Flow

https://guides.github.com/introduction/flow/



Other Git workflows:
https://www.atlassian.com/git/tutorials/comparing-workflows

Cloning

Each copy of the repo is a complete, fully-functional copy. That's why it is called *distributed* version control: there's no central version. Control is distributed among all of the versions.

The `git clone` command copies the remote repository to your local machine.

You can search for and clone a repository from GitHub using the **Git: Clone** command in the **Command Palette (Ctrl+Shift+P)** or by using the **Clone Repository button** in the Source Control view (available when you have no folder open).

https://code.visualstudio.com/docs/editor/github

Forking

“A **fork** is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.

Most commonly, forks are used to either propose changes to someone else’s project or to use someone else’s project as a starting point for your own idea.”

GitHub Help: https://docs.github.com/en/get-started/quickstart/fork-a-repo

Pull Requests

“**Pull requests** let you tell others about changes you’ve pushed to a repository on GitHub. Once a pull request is opened, you can discuss and review the potential changes with collaborators and add follow-up commits before the changes are merged into the repository.”

https://docs.github.com/en/github/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests

Git in the Terminal

`git init` = initialize new repository (a hidden directory called .git)

`git add .` = add everything in directory, i.e. ‘staging’ (note the period!)

`git commit -m "NDA - initial master commit"` = commit with message

`git log` = view the history of all changes made to repository

`git branch input-system` = create new branch named "input-system"

`git branch` = display a list of existing branches

`git checkout -b input-system` = “check out,” i.e., switch to, a branch

`git merge input-system` = merge branch into main branch

`git status` = where we’re at, what’s going on

Git Command Tutorial

http://git-scm.com/docs/gittutorial

I want to try something that might or might not work, but I don't want to mess up my work in progress.

Simple and "unprofessional" approach:

- Make another clone of the team's public repo (or just copy the whole dang folder).
- Make your experimental changes, *and maybe don't even commit them*.
- If/when they work, use the git diff viewer to figure out which changes were essential.
- Apply those changes in your main project using git.
- Delete your local copy.

Complex and "professional" approach:

- Optionally stash in-progress work.
- Create a feature branch with git.
- Make your changes, committing incremental changes and pushing your branch to the team's public repo.
- Once it works, submit a pull request to suggest this branch be merged into the main/master branch.
- Delete your local branch and sync the deletion with the public repo.

Use the right tool for the job

Going to the fridge to look for a snack?

Going to the FIFA World Cup?
Sure, put on your professional-grade technical footwear for optimal performance
(but please don't walk on the hardwood floor in the house)

Cooking analogy

Home cooking for your friends and family is different from professional catering for paying clients. Adopting tricks from the professionals can make you a better cook, but be wary of anything that threatens to stop you from successfully putting dinner on the table.

You need to deliver food (/an interactive experience CMPM/ARTG 120). You need to see what people think of it, tweak the recipe, and test your variation. You don’t need to pass a food safety inspection, build your company’s reputation, or win reviews from critics.

What About Assets

git is great at version control for text, but not so great for binary files. In general, we want track changes to our source, not our assets. To do so, we use a `.gitignore` file. This file is a set of rules that tells git which files to ignore before a commit. Be sure to do this first!

You can manage assets different ways:

- ➔ Separately (Dropbox, Google Drive, etc.) and copy the final versions in
- ➔ Using Git LFS (Large File Storage)
- ➔ Using a different VCS (such as Plastic or Perforce)

Asset version control

The what and why of commercial version control systems (VCSs) like Perforce: https://www.perforce.com/solutions/version-control

Note key phrases:

- “asset management” (not just code)
- “code review” (not just code storage)
- “automated builds”
- “automated testing”
- “continuous deployment”
- “center of your [intellectual property] universe”
- “identity and access management”
- “governance and regulatory needs”

Git's original use case

Supporting development of the linux kernel: https://github.com/torvalds/linux

Assumptions:

- ➔ Repo contains small text files (not large binary assets)
- ➔ Contributors from different organizations want to work in public without requiring access to shared resources (versus teams that want to guard IP, prevent leaks, or already have ways of sharing other design materials).

Git isn't a great match for videogame development, but it is a good match for less asset-heavy or IP-restricted things you might also do (web or desktop/mobile app development). At a AAA studio, you might use Perforce. (Or Plastic. Or SVN. Or Git.)

Merge tools

https://meldmerge.org/

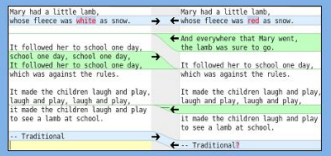
Meld

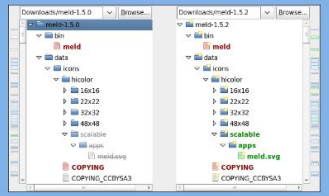
Get it | News | Features | Help | Wiki | Development

What is Meld?

Meld is a visual diff and merge tool targeted at developers. Meld helps you compare files, directories, and version controlled projects. It provides two- and three-way comparison of both files and directories, and has support for many popular version control systems.

Meld helps you review code changes and understand patches. It might even help you to figure out what is going on in that merge you keep avoiding.





Features

- Two- and three-way comparison of files and directories
- File comparisons update as you type
- Auto-merge mode and actions on change blocks help make merges easier
- Visualisations make it easier to compare your files
- Supports Git, Bazaar, Mercurial, Subversion, etc.
- …and more

Meld is licensed under the GPL v2, except as noted.

Getting it

Meld is packaged for most Linux/Unix distributions, including Fedora, Ubuntu, and Suse. Unless you want the absolutely latest version, you should install Meld through your package manager.

Windows users should download the MSI, or for older releases, check out the Meld installer project.

On OS X, Meld is not yet officially supported. For pre-built binaries, these OS X builds are the best option. You can also get Meld from MacPorts, Fink or Brew; none of these methods are supported.

You can also run Meld without installing it. Just extract the archive and run `bin/meld` from the archive folder.

Source:	Meld 3.20.1	31 March 2019
Windows:	Meld 3.20.0	6 January 2019

Note: The 3.20 Windows build uses a new build chain. If you experience issues, please use Meld 3.18.3.

Requirements

- Python 3.3
- GTK+ 3.14
- GLib 2.36
- PyGObject 3.14
- GtkSourceView 3.14
- pycairo

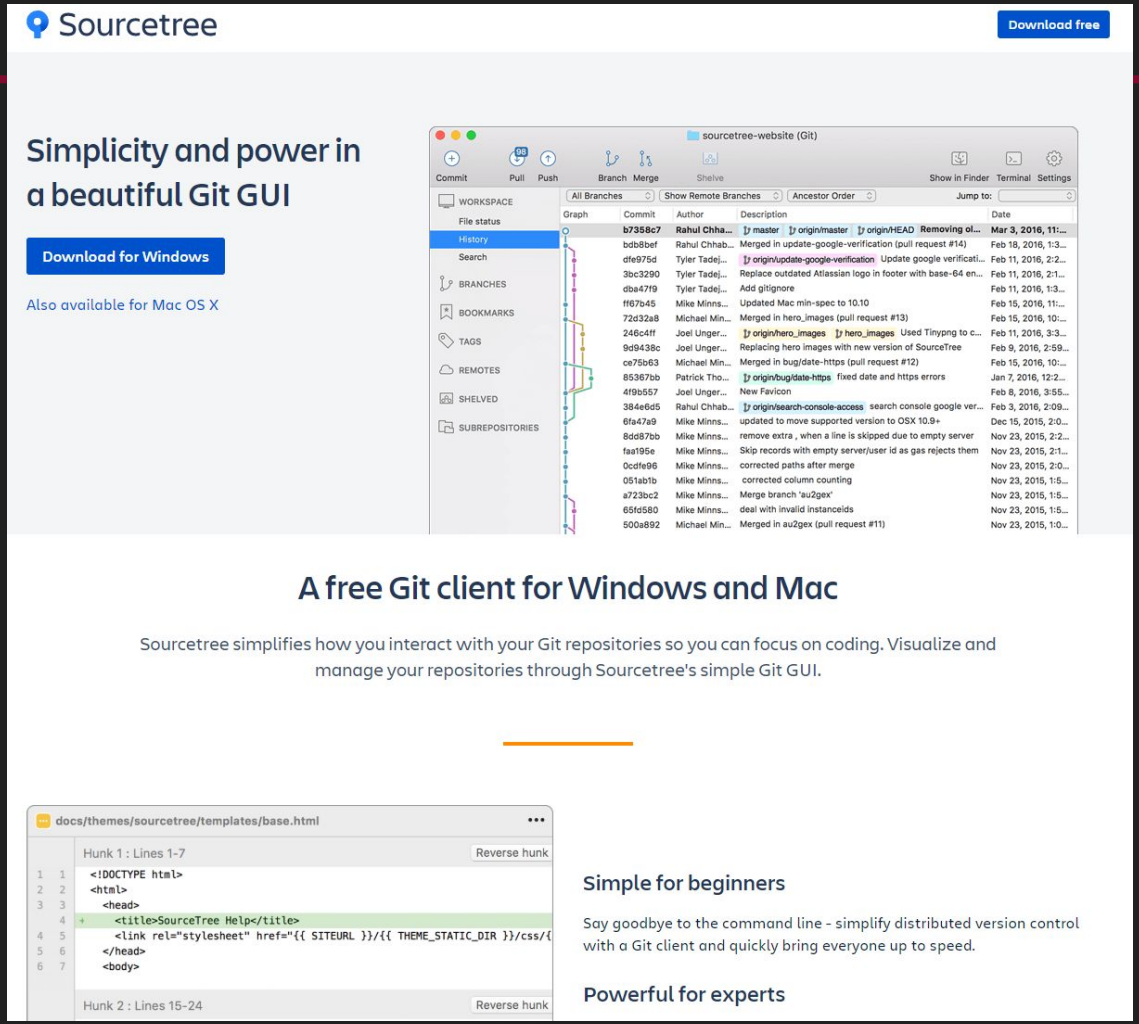
Copyright © 2011-2012 Kai Willadsen Hosted by GitHub

Git tools

https://www.sourcetreeapp.com/

https://www.gitkraken.com/

https://git-scm.com/downloads/guis



GitHub and Alternatives

https://github.com/

https://bitbucket.org/product

https://about.gitlab.com/product/source-code-management/

Break

Debugging

A software bug (in vague terms)

When you have a pretty clear expectation for how the system should behave, but it does something different.

Examples:

- I thought my code would run, but it doesn't.
- My code works for me, but it doesn't work for you.
- My code worked yesterday, but it no longer works today.
- I thought my code would finish running, but it kept going.

That one time a program in the Google Music system deleted everyone's mp3s: https://sre.google/sre-book/data-integrity/

92

9/9

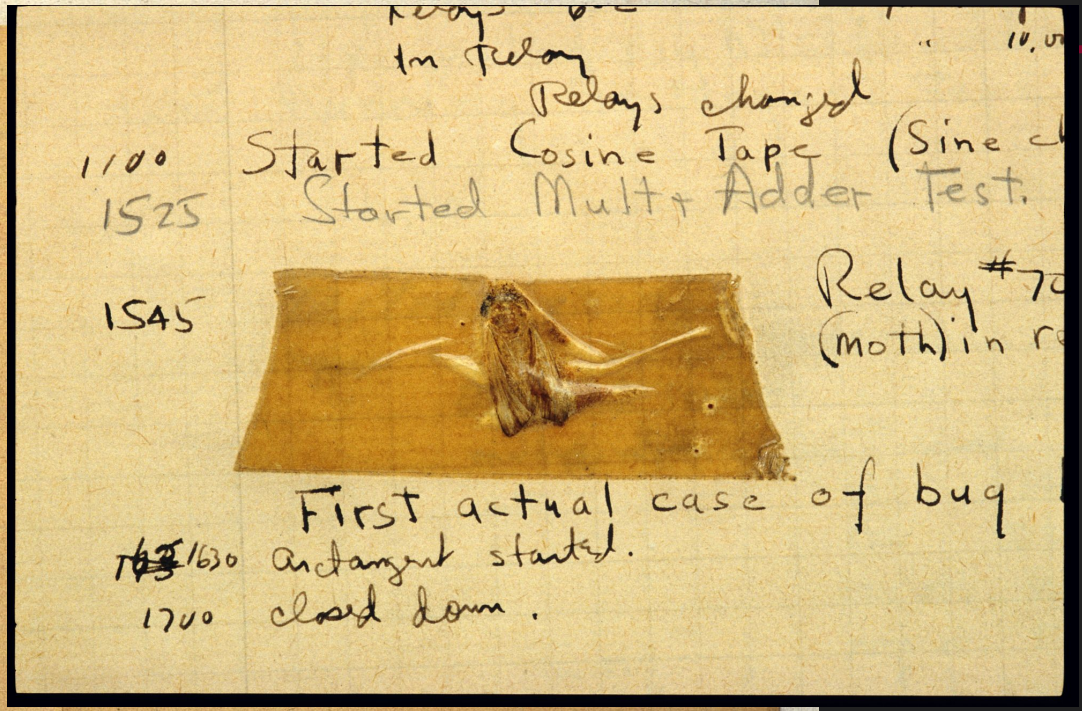
0800 Arctan started
1000 " stopped - arctan ✓
13 ⁵/₈ (034) MP - MC
033 PRO 2
correct

{ 1.2700 9.037 847 025
9.037 846 985 correct
4.615725059(-2)
2.130476415
2.130676415

Relays 6-2 in 033 failed special speed test
In Relay
Relays changed
10,000 test.

1100 Started Cosine Tape (Sine check)
1525 Started Multy Adder Test.
1545 Relay #70 Panel F
(moth) in relay.

First actual case of bug being found.
~~1645~~ 1630 Arctangent started.
1700 closed down.



See also: Grace Hopper and the notion of "automatic programming"

https://americanhistory.si.edu/collections/search/object/nmah_334663

Expert programmers

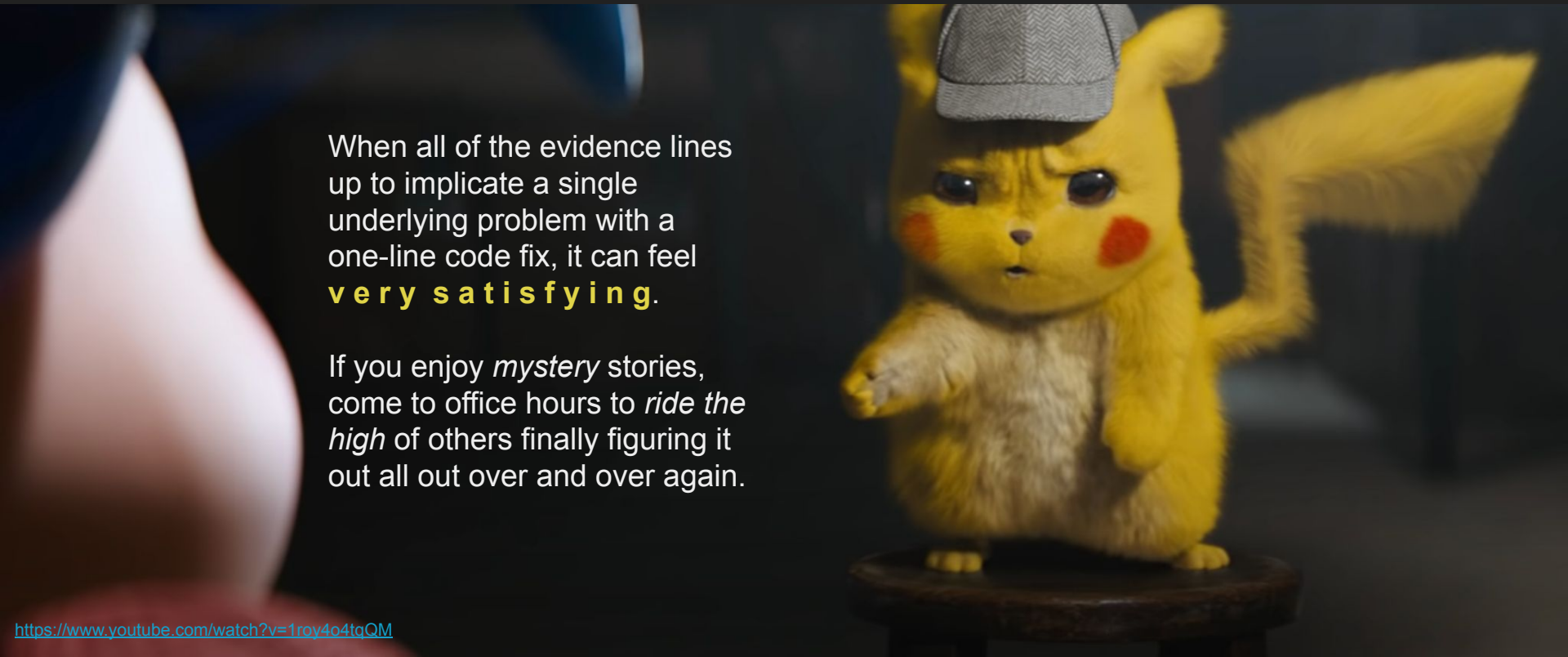
~~The know that bugs waste time, so they simply don't introduce and new ones.~~

Expert programmers constantly introduce bugs. The difference between them and novice programmers is often how quickly the can *diagnose* and *fix* the underlying problems.

If you are 95% accurate (very high!) in expressing each code token (a variable name, a brace, some indentation), there's a >50% chance you screw up before you've typed 14 tokens in a row.

Expert programmers are also less likely to attempt bug-prone designs, but the morale effect of getting quickly unstuck is maybe a bigger deal.

The thrill of debugging



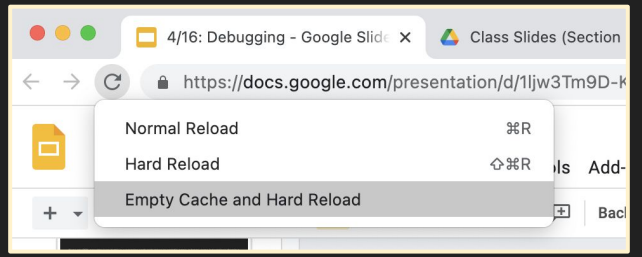
When all of the evidence lines up to implicate a single underlying problem with a one-line code fix, it can feel **v e r y s a t i s f y i n g**.

If you enjoy *mystery* stories, come to office hours to *ride the high* of others finally figuring it out all out over and over again.

https://www.youtube.com/watch?v=1roy4o4tqQM

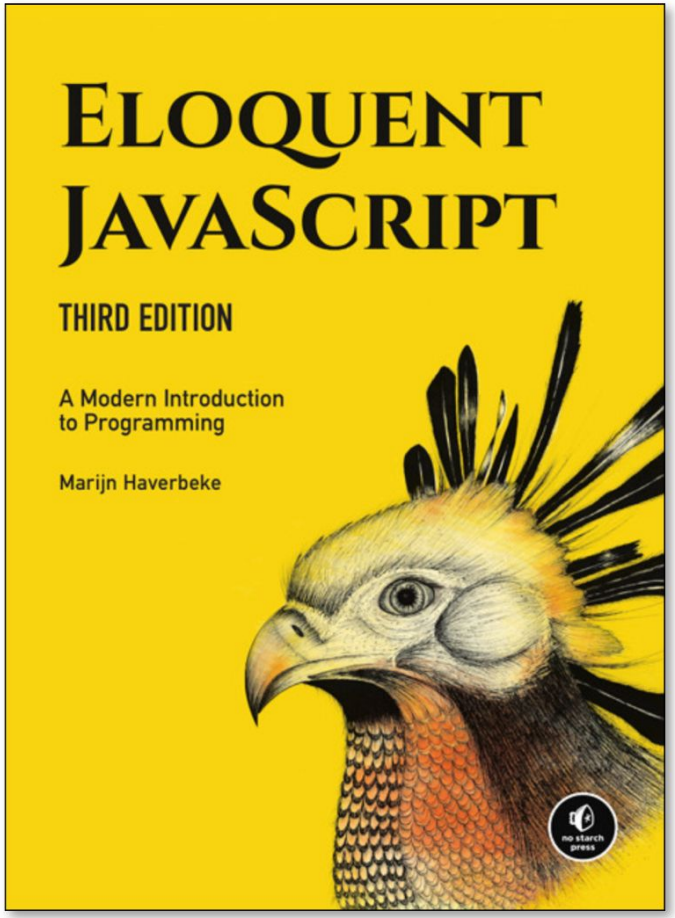
Common villains

- Smart quotes: “ vs "
- Capitalization: **Assets** vs. **assets**
- Semicolons: **;**
- Keyword/identifier spelling: **return** vs. **retrun**
- Not saving your file: **⌘+s** or **Ctrl+S**
- Drawing graphics offscreen:
- Forgetting scope: **p1Score** vs. **this.p1Score**
- The code that is running is different than you think...
- Viewing *cached* version of your page/assets:



Tools (and practices) help...

- **Type less buggy code.** [autocomplete]
- **Spot bugs as you type them.** [syntax highlighting/wigglies]
- **Expose you to feedback about executing.** [consoles / logging systems]
- **Build expectations for normal behavior.** [docs, peers, web search engines]
- **Monitor systems on which your game depends.** [network tracers, etc.]
- **Gather evidence about buggy behavior.** [graphical debuggers]
- **Compare and recover past working versions.** [version control (git)]
- **Understand which code is running when.** [profilers]
- **Approximate deployment conditions.** [screen and network simulators]
- ...



ELOQUENT JAVASCRIPT
3RD EDITION (2018)

CONTENTS

← https://eloquentjavascript.net/08_error.html

Break

Introducing the Endless Runner

Endless Runner

Due July 6 (next Tuesday).

Starting from **blank** files (and a recently downloaded copy of phaser.js), your **team** should make and release a complete, playable game of your own design.

It should be recognizable as an **endless runner** game, but almost all of the creative decisions are up to you.

What is an Endless Runner? Help me fill this in.

What makes it **endless**?

No border

It doesn't end

Can technically be played forever

Like temple run

Can go in a direction forever

Automatic forward movement

Can not go back

What makes it a **runner**?

Increasing speed

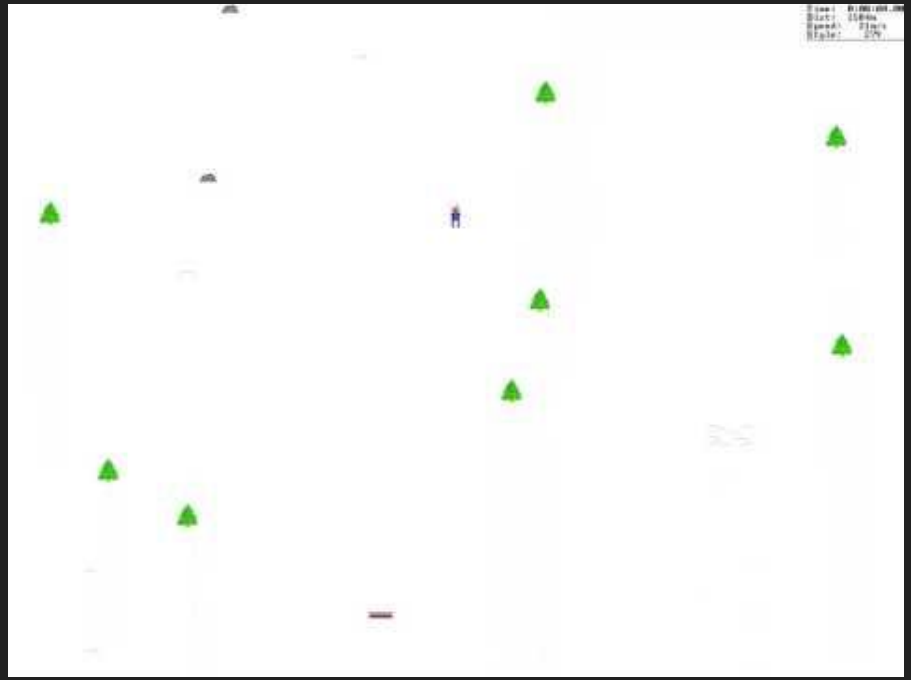
Obstacles

Based around moving forward

(like a monster catching you)

Travelling as far as possible

SkiFree and *Canabalt*



Nathan's *Paddle Parkour P3*

Let's check out (git clone) Nathan's example endless runner demo built on the Phaser 3 game framework:
https://github.com/nathanaltice/PaddleParkourP3

Endless: enemy paddle swarm of increasing complexity

Runner: bounce your paddle, dodging enemies

Missing: Sprite atlas and some other requirements

Included: credits in main.js

What's required by the assignment?

Let's check the details on Canvas...

Assignment:

https://canvas.ucsc.edu/courses/44176/assignments/256904

Groups:

https://canvas.ucsc.edu/courses/44176/groups

Teamwork

- **Everyone** on the team needs to contribute to the git repository.
- Outside of that, your contribution to the team can include research into how to do things.
- Explain what you are doing to each other.
- If you are a more experienced programmer, part of your job will be **helping your other team members learn!**

Speedrunning the Endless Runner

- You've already seen a version of everything you'll need for this.
- The goal is to give you practice at making your own versions.
- I'm going to be explaining the concepts as we go along, but you can anticipate what you need and do your own research.
- Problem/Solution ordering:
  - Encounter the problem and grapple with it before having the solution explained

Resources for the Endless Runner

Sprite Sheets and Texture Atlases

https://www.codeandweb.com/texturepacker/tutorials/how-to-create-sprite-sheets-for-phaser3

Endless Scrolling with tileSprite

https://www.thepolyglotdeveloper.com/2020/08/continuous-side-scrolling-phaser-game-tile-sprites/

https://docs.idew.org/video-game/project-references/phaser-coding/tilesprite-scrolling

https://blog.ourcade.co/posts/2020/add-pizazz-parallax-scrolling-phaser-3/

What is different between these examples? What are the similarities?

Break

Scenes

Multiple Scenes in Nathan's *AVeryCapableGame* (Phaser)

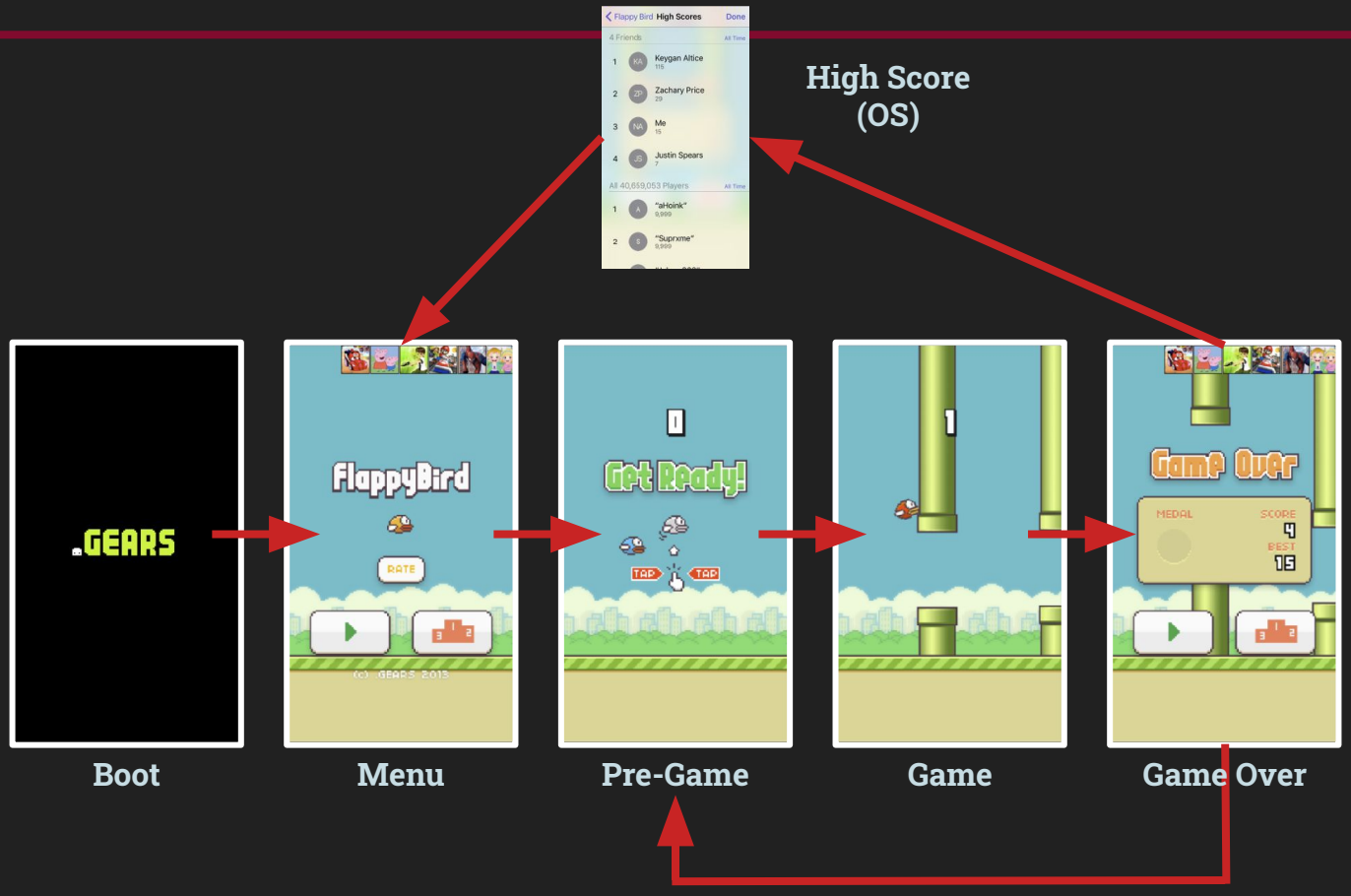
1. Let's **clone** Nathan's game from GitHub: https://github.com/nathanaltice/AVeryCapableGame
2. How does one scene transition to another?
3. How does one scene share player data with another?
4. How should one put on scene *on top* of another, like to build an inventory screen used *during* main gameplay?

Multiple Scenes in Nathan's *AVeryCapableGame* (Phaser)

- Let's **clone** Nathan's game from GitHub: https://github.com/nathanaltice/AVeryCapableGame
- How does one scene transition to another?
  - `this.scene.start(SomeSceneClass)`
- How does one scene share player data with another?
  - `this.scene.start(SomeSceneClass , dataObject)`
- How should one put on scene *on top* of another, like to build an inventory screen used *during* main gameplay?
  - `this.scene.launch(SomeSceneClass)`

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

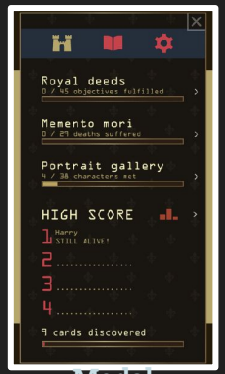

Flappy Bird High Scores
Done
Keygan Altice
Zachary Price
Me
Justin Spears
"aHoink"
"Suprxme"
High Score
(OS)
.GEARS
FlappyBird
RATE
Get Ready!
TAP
TAP
Game Over
MEDAL
SCORE
BEST
Boot
Menu
Pre-Game
Game
Game Over

Credits

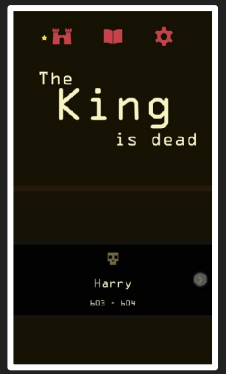
Title

Spawn

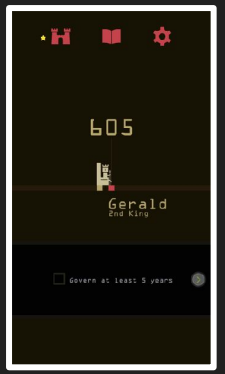
Play



Modal Menu



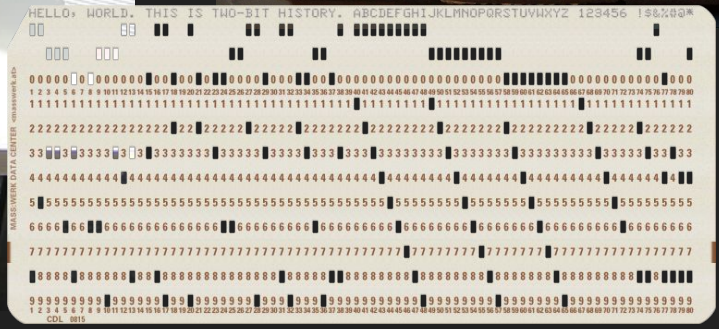
Game Over



Legacy

Game Loops

Punchcards for input/output



https://twobithistory.org/2018/06/23/ibm-029-card-punch.html

Batch processing

The “central processing unit” (CPU)

Put your manually-punched input cards on the slide here

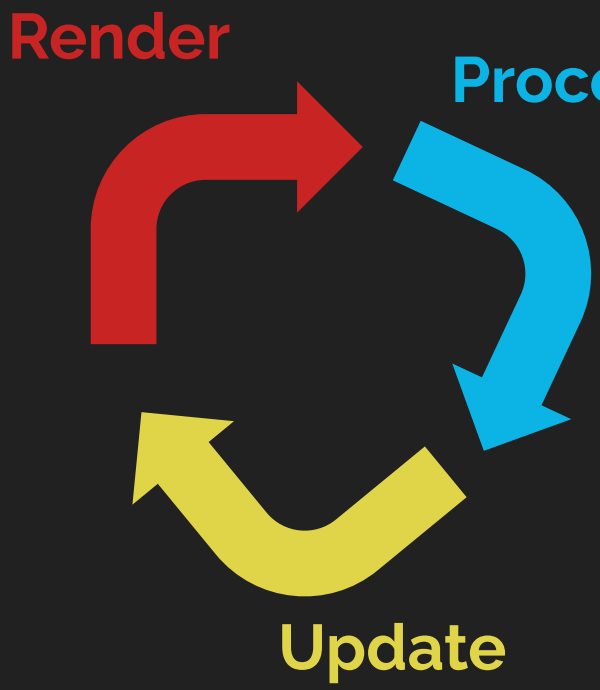
See punched+printed output cards here several hours later

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

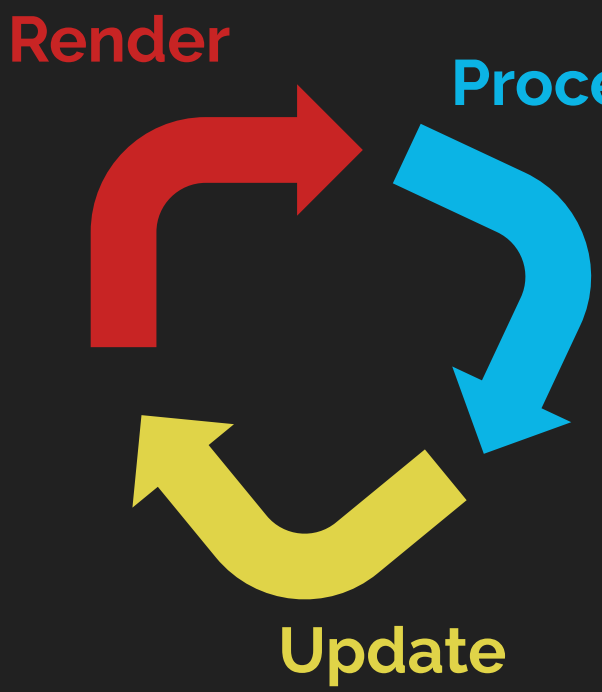
Game Programming Patterns, p. 304

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?

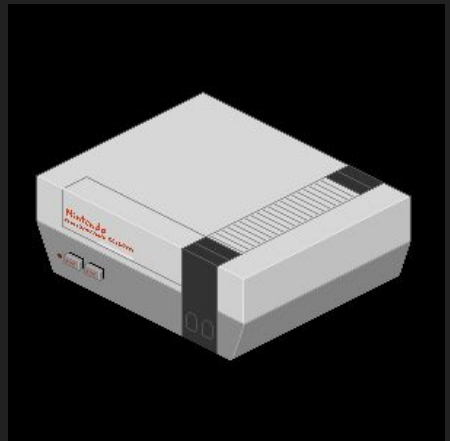
CPU speed, memory resources, screen refresh rate, operating system preemption, etc.

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

For non-game web dev, you might want `setTimeout()` instead.

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:

1. A first attempt
2. Timing problems
3. Physics problems
4. A solution
5. Panic! Spiral of death

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

Physics

Hamiltonian mechanics

Generalized positions

Generalized momenta

$$\frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t} = -\frac{\partial \mathcal{H}}{\partial \boldsymbol{q}} \quad , \quad \frac{\mathrm{d}\boldsymbol{q}}{\mathrm{d}t} = +\frac{\partial \mathcal{H}}{\partial \boldsymbol{p}}$$

Will deeper understanding of calculus and linear algebra lead to better game programming skills?
It'll make some things more intuitive.

Can I be a great game programmer without any calculus or linear algebra?
Certainly!

Game physics in Phaser

Option 1 (no physics): *I'll implement the laws of the universe myself, thanks.*

Option 2 (Arcade Physics): *Give me a starting point for making games with collision and movement in the style of 2D platformer games.*

Option 3 (Matter Physics): *Give me rotational inertia, constrained joints, and lots of linear algebra to think about while I debug.*

No (built-in) physics

This is how we implemented RocketPatrol.

Phaser's **sprites** are like **removable stickers**:

- They have a position (obj.x, obj.y)
- They have a shape (obj.width, obj.height)
- They can be moved:
  - obj.x = game.config.width / 2;

Objects don't automatically **move** over time -- we move/teleport them manually using code in update().

Stickers safely overlap. To make things look like they **collided**, we continually checked for overlap in update().

https://stickerplus.com.au/removable-sticker

Arcade physics

With arcade physics, objects have:

- Position: obj.body.x
- Velocity: obj.body.velocity.x
- Specific collision shapes (circle w/ radius)
- Specific collision responses (immovable, bounce, etc.)

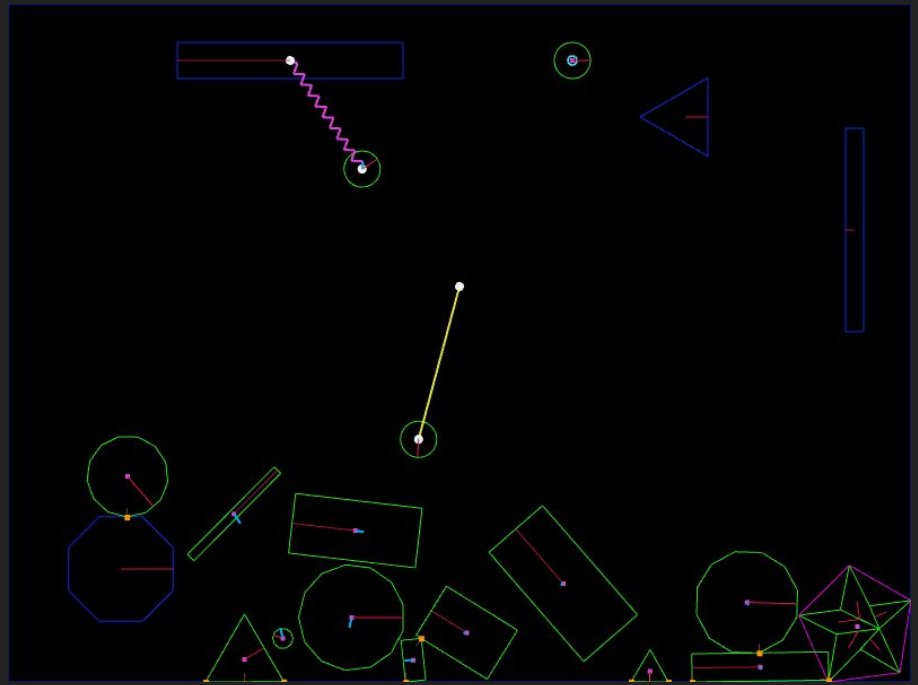
Objects in motion stay in motion (moving with velocity that changes with the scene's gravity) unless their properties are manually changed.

Matter physics

With Matter physics, objects may have:

- Sub-objects
- Continuous rotation / inertia
- Springs/constraints of different stiffness
- ...

Unless rotation or spring physics are needed for your game design, stick with Arcade physics in this class.



https://phaser.io/examples/v3/view/physics/matterjs/debug-options

Complex physical constraints in *Garry's Mod*

Nathan's MovementStudies

Let's clone this project and play with each of the numbered demo scenes. (cheat codes!)

Demo notes:

- Define physics using the game config object. (note `debug: true`)
- Concept: arcade bodies
- this.add.sprite → this.physics.add.sprite
- this.cloud01.body.setAllowGravity(false)
- this.physics.add.collider(a, b)
- this.alien.setVelocityX(...) or this.alien.body.velocity.x
- this.physics.world.wrap(obj, width)
- this.group.add()

Collisions

Visual game objects versus physical game objects

https://stickerplus.com.au/removable-sticker

The sticker stays put until *you* move it yourself.

https://www.firefliesandmuspies.com/mess-free-sticker-rocks/

When you drop or throw the rock, *nature* moves, the sticker moves along with it.

P.GameObject

Extends

- Phaser.Events.EventEmitter

P.GO.Sprite

Extends

- Phaser.GameObjects.GameObject
- Phaser.GameObjects.Components.Alpha
- Phaser.GameObjects.Components.BlendMode
- Phaser.GameObjects.Components.Depth
- Phaser.GameObjects.Components.Flip
- Phaser.GameObjects.Components.GetBounds
- Phaser.GameObjects.Components.Mask
- Phaser.GameObjects.Components.Origin
- Phaser.GameObjects.Components.Pipeline
- Phaser.GameObjects.Components.ScrollFactor
- Phaser.GameObjects.Components.Size
- Phaser.GameObjects.Components.TextureCrop
- Phaser.GameObjects.Components.Tint
- Phaser.GameObjects.Components.Transform
- Phaser.GameObjects.Components.Visible

P.P.A.Sprite

Extends

- Phaser.GameObjects.Sprite
- Phaser.Physics.Arcade.Components.Acceleration
- Phaser.Physics.Arcade.Components.Angular
- Phaser.Physics.Arcade.Components.Bounce
- Phaser.Physics.Arcade.Components.Debug
- Phaser.Physics.Arcade.Components.Drag
- Phaser.Physics.Arcade.Components.Enable
- Phaser.Physics.Arcade.Components.Friction
- Phaser.Physics.Arcade.Components.Gravity
- Phaser.Physics.Arcade.Components.Immovable
- Phaser.Physics.Arcade.Components.Mass
- Phaser.Physics.Arcade.Components.Pushable
- Phaser.Physics.Arcade.Components.Size
- Phaser.Physics.Arcade.Components.Velocity
- Phaser.GameObjects.Components.Alpha
- Phaser.GameObjects.Components.BlendMode
- Phaser.GameObjects.Components.Depth
- Phaser.GameObjects.Components.Flip
- Phaser.GameObjects.Components.GetBounds
- Phaser.GameObjects.Components.Origin
- Phaser.GameObjects.Components.Pipeline
- Phaser.GameObjects.Components.ScrollFactor
- Phaser.GameObjects.Components.Size
- Phaser.GameObjects.Components.Texture
- Phaser.GameObjects.Components.Tint
- Phaser.GameObjects.Components.Transform
- Phaser.GameObjects.Components.Visible

A custom marshmallow object

```
class Marshmallow extends Phaser.Physics.Arcade.Sprite {
      constructor(scene, x, y, texture, frame) {
            super(scene, x, y, texture, frame);
            this.temperature = 68;
            this.burned = false;
      }

      roast() {
            this.temperature += 5;
      }

      update() {

            if (this.temperature > 350) {
                  this.burned = true;
            }

            // slowly return to room temperature
            this.temperature = (this.temperature - 68) * 0.95 + 68;
      }
}
```

Collision Handling

polling-based

vs

event-based

Hey *nature*, can you tell me when something interesting happens?

```
let player = ...;

let enemy1 = ...;
let enemy2 = ...;

let enemyGroup = this.physics.add.group([enemy1, enemy2]);

this.physics.add.collider(player, enemyGroup, (p,e) => {
    console.log('Player collided with enemy: ', e);
});
```

This is event-based collision handling.
We saw polling-based collision handling in Rocket Patrol.

Rule: Every frame, the marshmallow will be roasted by each fire particle that it touches.

```
update() {
    ...

    this.physics.world.collide(
        marshmallow,
        fireParticles,
        this.touchedFire);
}

touchedFire() {
    marshmallow.roast()
}
```

This is polling-based collision handling, using Arcade Physics to check geometry for us.

Do I have to make a new class for every kind of physical object?

No. You only need a new **class** if that kind thing needs special state (variables) or behavior (methods) of its own.

Let's check out the **balls** and **crates** in https://phaser.io/examples/v3/view/physics/arcade/sprite-vs-multiple-groups#

Let's revisit PaddleParkourP3

What objects are visible on screen?

Are there any custom (new class) physical Sprites?

Is collision handled by polling or event callbacks?

Do barriers live forever off-screen?

How does .update() get called on barrier objects??

Objects

Input and Movement

State Machines

Cameras

Bonus Slides

Demo notes follow

(watch the recording for reference; these are just notes for what to show live)

Type less buggy code.

- Use autocomplete to spell identifiers.
- Read your code in your head (or even aloud) as you type.
- Don’t rush (very little programming time is spent typing anyway; most is reading/thinking/watching/debugging).

Spot bugs as you type them.

- Get someone to watch you! / Watch other people!
- Monitor the colors of the syntax highlighting.
- Pay attention to wiggly underlines.
- Make sure your tools give you feedback (colors/wigglies).

Expose yourself to feedback about execution.

- Keep your browser console open any time you are changing your code (and maybe even while watching a playtester).
- Use console.log(...) and console.assert(...) to put more messages in the console.
- Read location indicators like “main.js:42”, or just click to jump to a potential source of trouble. (In Chrome, you’ll then be the Sources devtools panel, which is not the same as VSCode.)
- Error messages are **clues** about problems (one bug might lead to 50 error message or zero), not problems by themselves.

Build expectations for normal behavior.

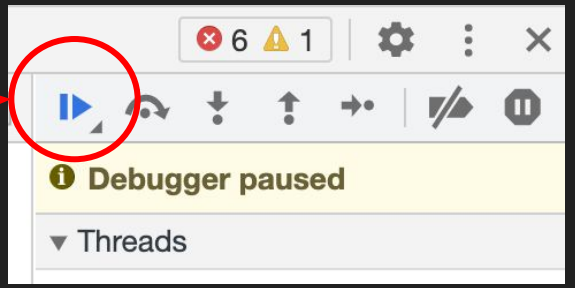
- Read online documentation for intended use of different API methods.
- Read **and edit** code examples made by others.
- Do web searches for error messages.
- Talk through your code while others are watching, confirming that the good parts of the code are indeed good (checking reasoning; don't be right for the wrong reasons).

Monitor systems on which your game depends.

- Your game depends on some web server (and even parts of the wider internet) to correctly function!
- Check the Network panel of Chrome's devtools as you refresh the page.
  - Does the URL look right?
  - Does the file actually exist? (With the exact spelling and capitalization?)

(Actively) gather evidence about buggy behavior.

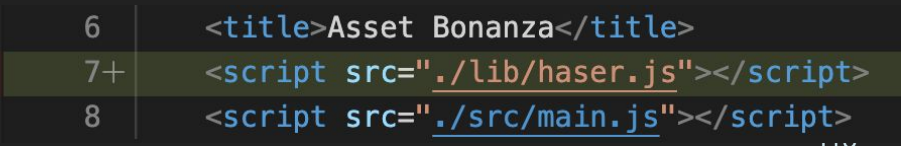
- Passive evidence gathering: old console.log(...) messages
- Active evidence gathering:
  - Slip a "`debugger;`" statement into some suspicious code (but also learn to use the Resume button)
  - In the graphical debugger, mouse-over variables of interest.
  - Add **logpoints** to existing code without restarting the game.



Metaphor: Cautiously feeling around in the dark versus turning on a flashlight and pointing it in the direction of the scary noise.

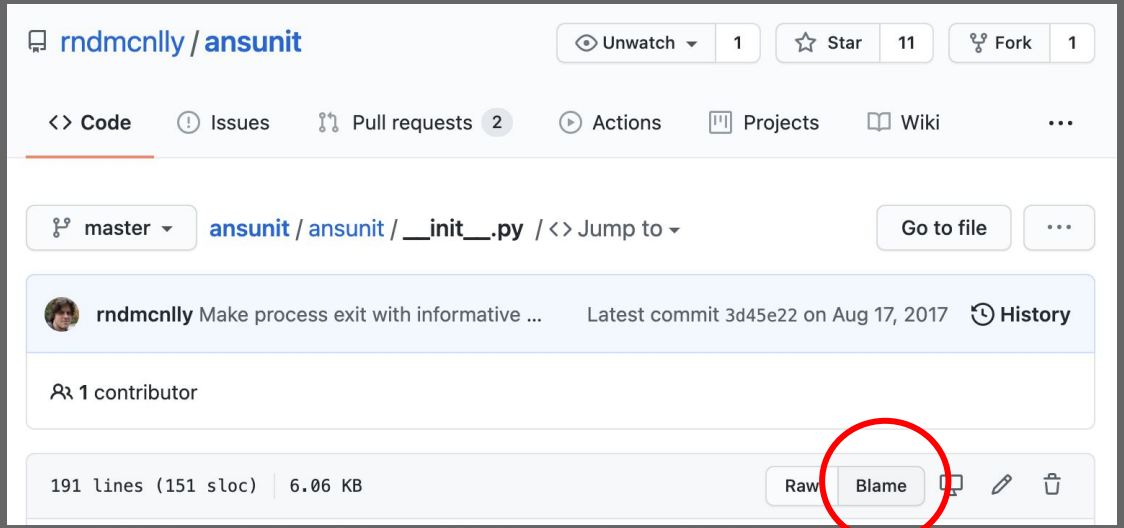
Compare and recover past working versions

- Use VSCode's Source Control tab review changes made since the last commit.
- "Stash" your current version; "checkout" the last working version to gather evidence of old behavior
- Use "tags" and "branches" to name and switch between specific versions of your game without losing the development history.
- ...



Aside: git blame

Git has a "blame" feature that can be used to (approximately) figure out who changed every line of code and what they were doing at the time. Use it to gather evidence of expected behavior -- don't actually use it to *blame* people for your problems.



https://github.com/rndmcnlly/ansunit/blob/master/ansunit/__init__.py

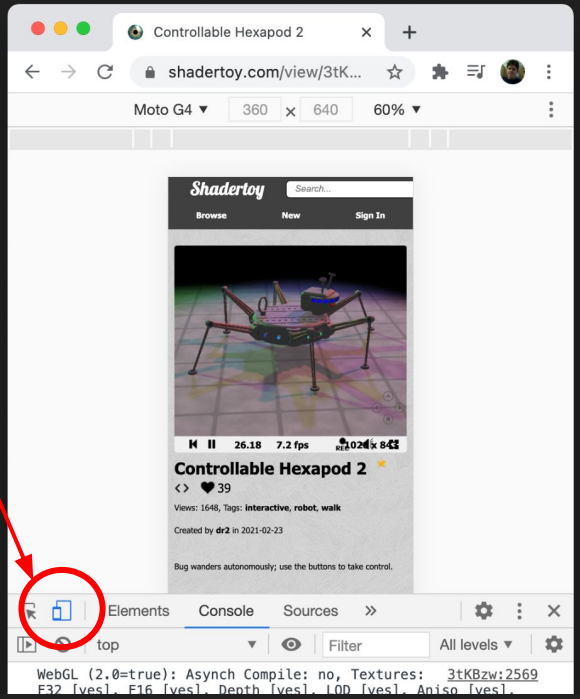
Understand what code is running when.

- Chrome devtool's Performance panel can be used to record a short execution sequence that you can graphically browse later.
- This is good for noticing the source of framerate slowdowns.
- Even when you don't care about performance, the profiler can give you a kind of "big picture" view of what happens while your game is running (sometimes useful for understanding code by others without reading all of it first).

Approximate deployment conditions.

- Use the Device Toolbar to simulate specific screen sizes or slow networks.

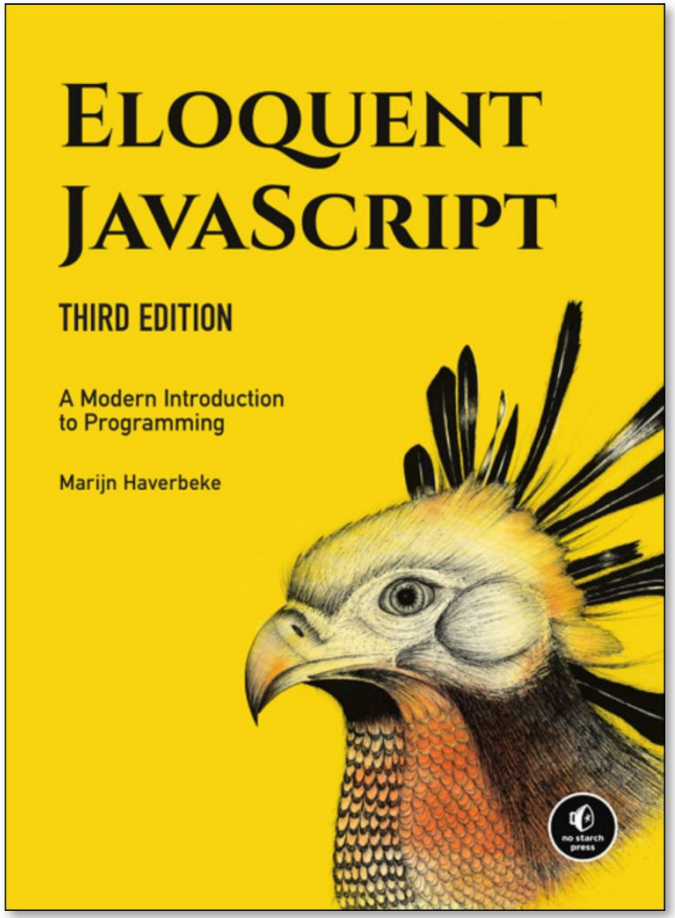
- Publish playable versions of your game with GitHub pages to get people to try your game on different operating systems, networks, browsers, etc.



Get machines to playtest your games (someday)

- Not (yet) built into browser devtools, but someday it might be.
- https://www.lambdatest.com/blog/monkey-testing-with-webdriverio/
- https://developer.android.com/studio/test/monkey
- Please use only *artificial* monkeys.

http://independentpress.cc/elon-musks-neuralink-shows-monkey-playing-video-game-with-mind/2021/04/10/



ELOQUENT JAVASCRIPT
3RD EDITION (2018)

CONTENTS

← https://eloquentjavascript.net/08_error.html