CMPM 120

Scenes & Physics

Schedule Overview*

6/22	Introduction
6/24	Programming Our First Phaser Game
6/29	Version Control & Debugging
7/1	Scenes, Loops, Physics
7/6	Input & State Machines
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

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.

Scenes

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

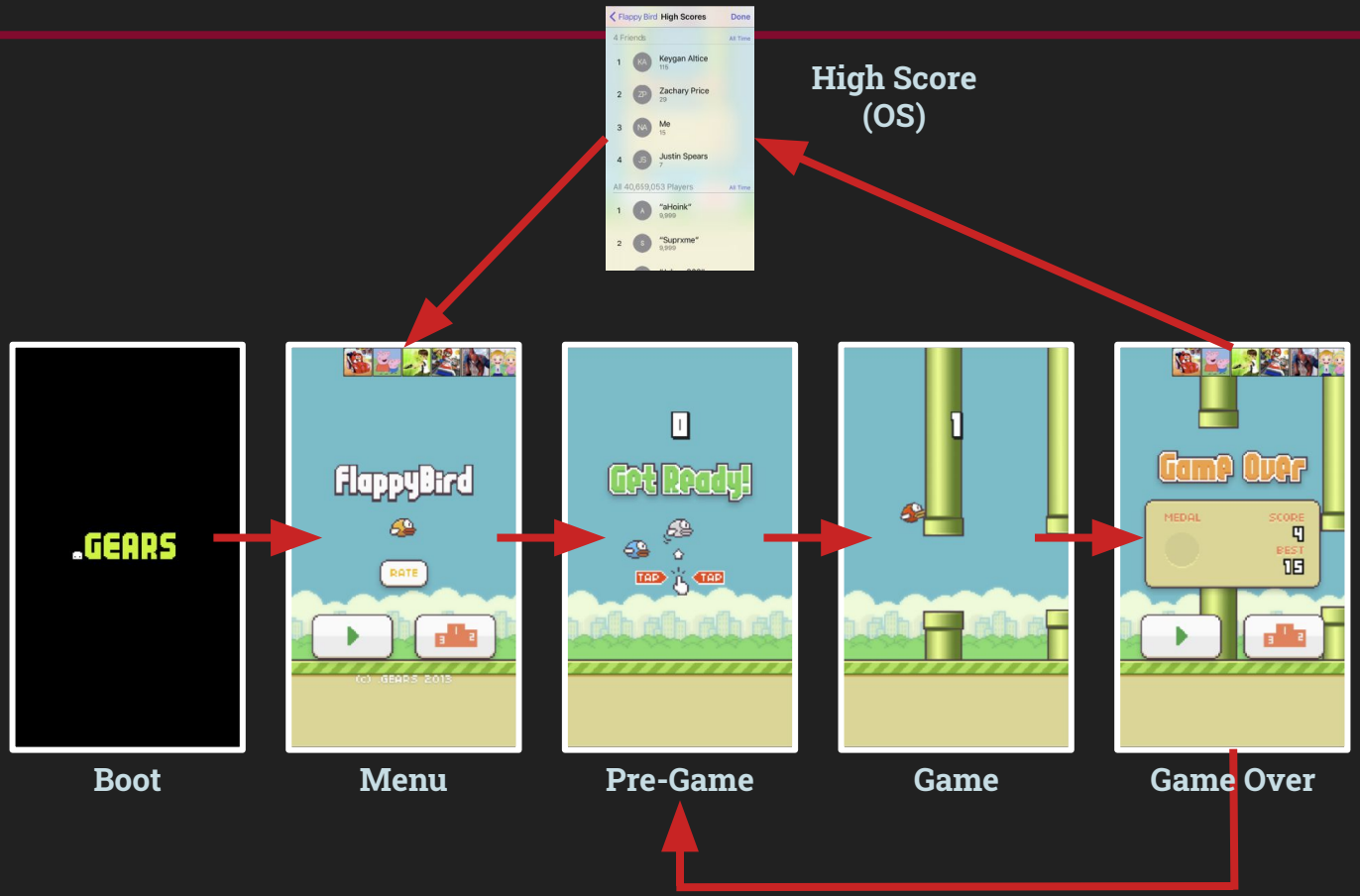
1. Let's **clone** Nathan's game from GitHub: https://github.com/nathanaltice/AVeryCapableGame
2. How does one scene transition to another?
3. How does one scene share player data with another?
4. How should one put on scene *on top* of another, like to build an inventory screen used *during* main gameplay?

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

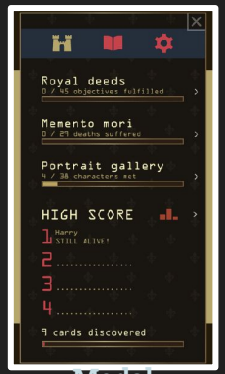

Flappy Bird High Scores
Done
4 Friends
All Time
1 Keygan Altice 115
2 Zachary Price 29
3 Me 15
4 Justin Spears 7
All 40,659,053 Players
1 "aHoink" 9,999
2 "Suprxme" 9,999
High Score
(OS)
.GEARS
FlappyBird
RATE
(c) GEARS 2013
0
Get Ready!
TAP
TAP
1
Game Over
MEDAL
SCORE
4
BEST
15
Boot
Menu
Pre-Game
Game
Game Over

Credits

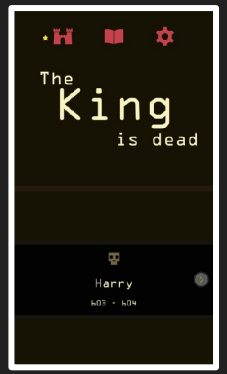
Title

Spawn

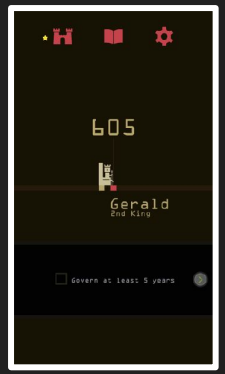
Play



Modal Menu



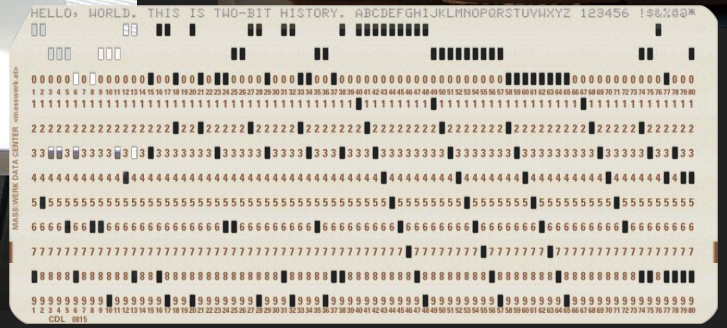
Game Over



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

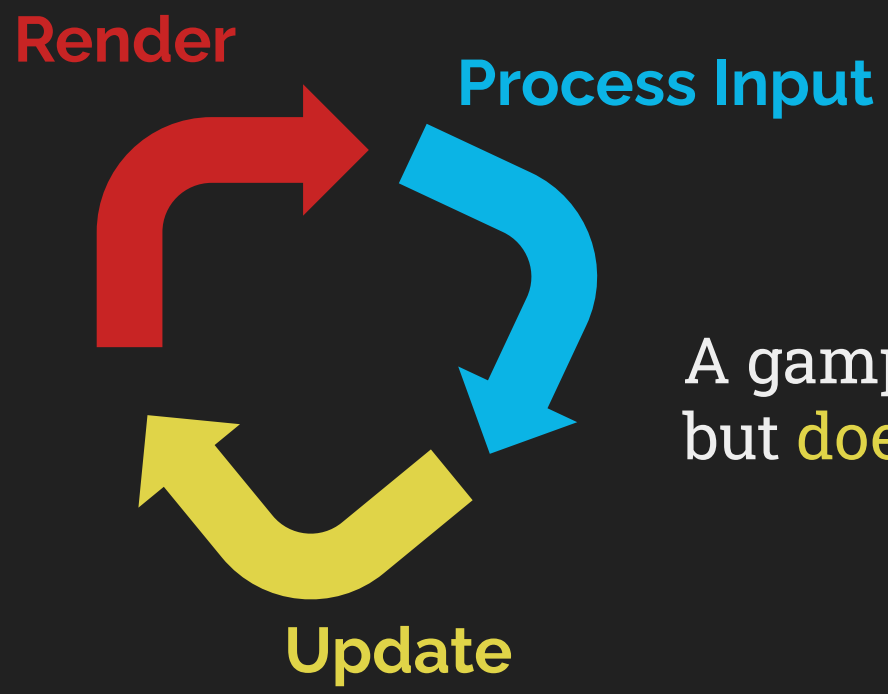
See punched+printed output cards here several hours later

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

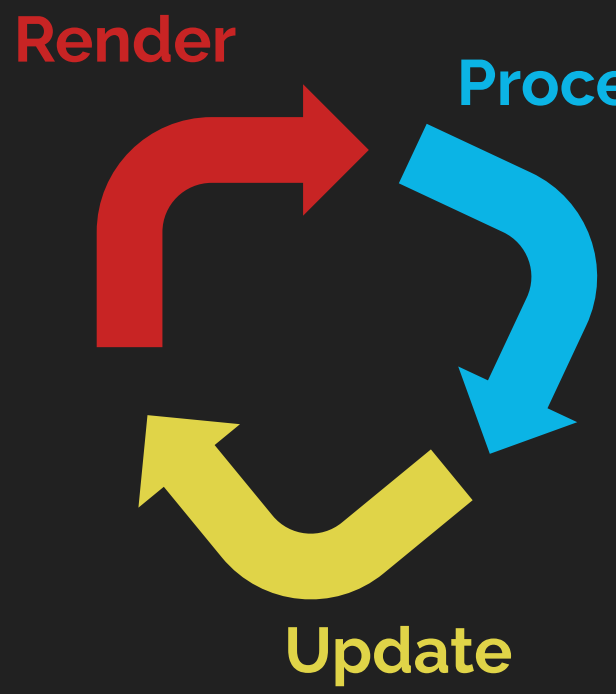
Game Programming Patterns, p. 304

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?

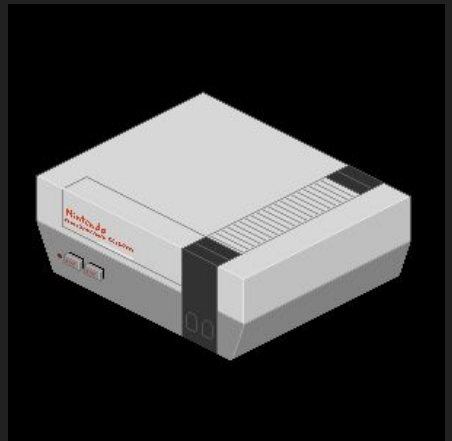
CPU speed, memory resources, screen refresh rate, operating system preemption, etc.

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

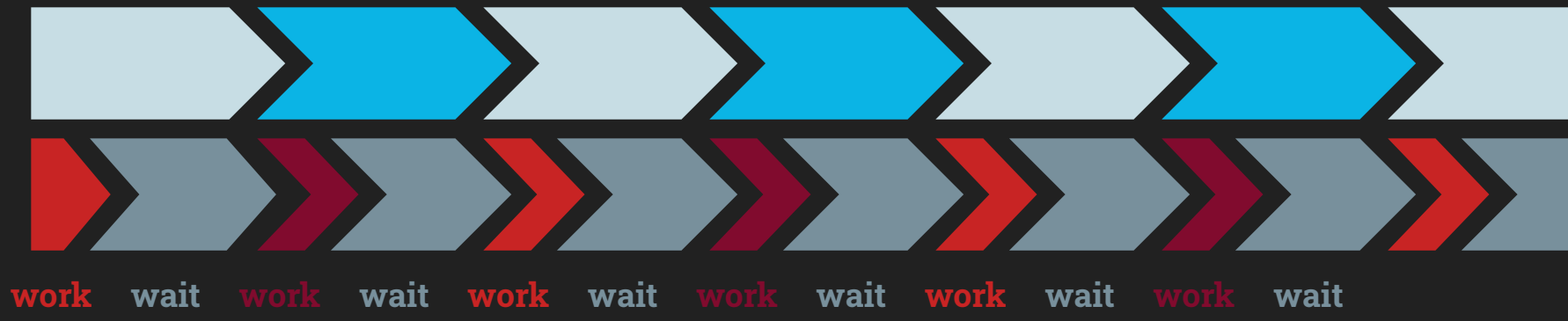
A callback function is a function passed into another function as an argument, which is then invoked inside the outer function to complete some kind of routine or action.

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.)

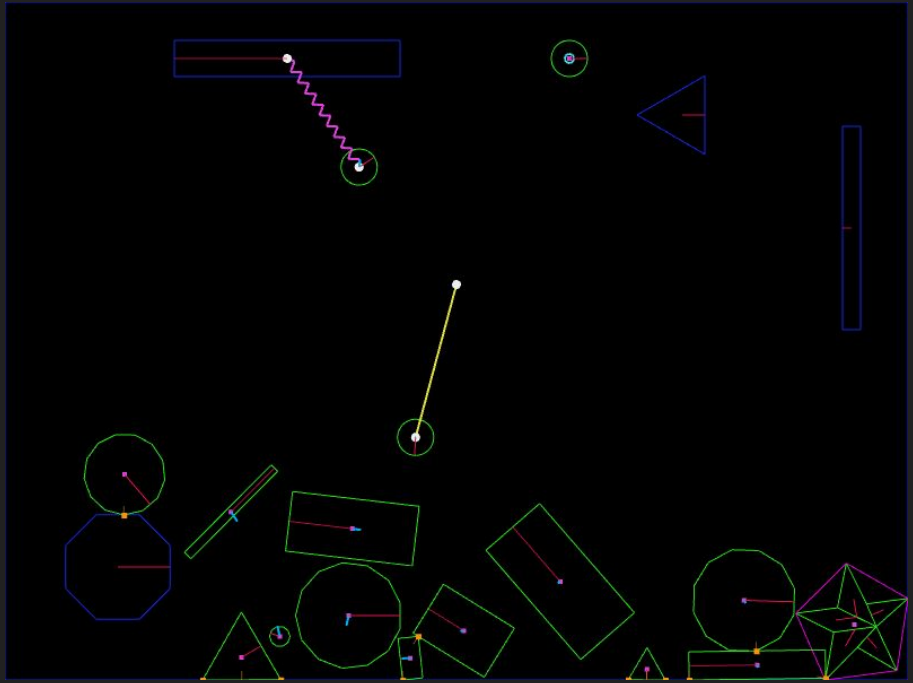
Objects in motion stay in motion (moving with velocity that changes with the scene's gravity) unless their properties are manually changed.

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.firefliesandhuspies.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