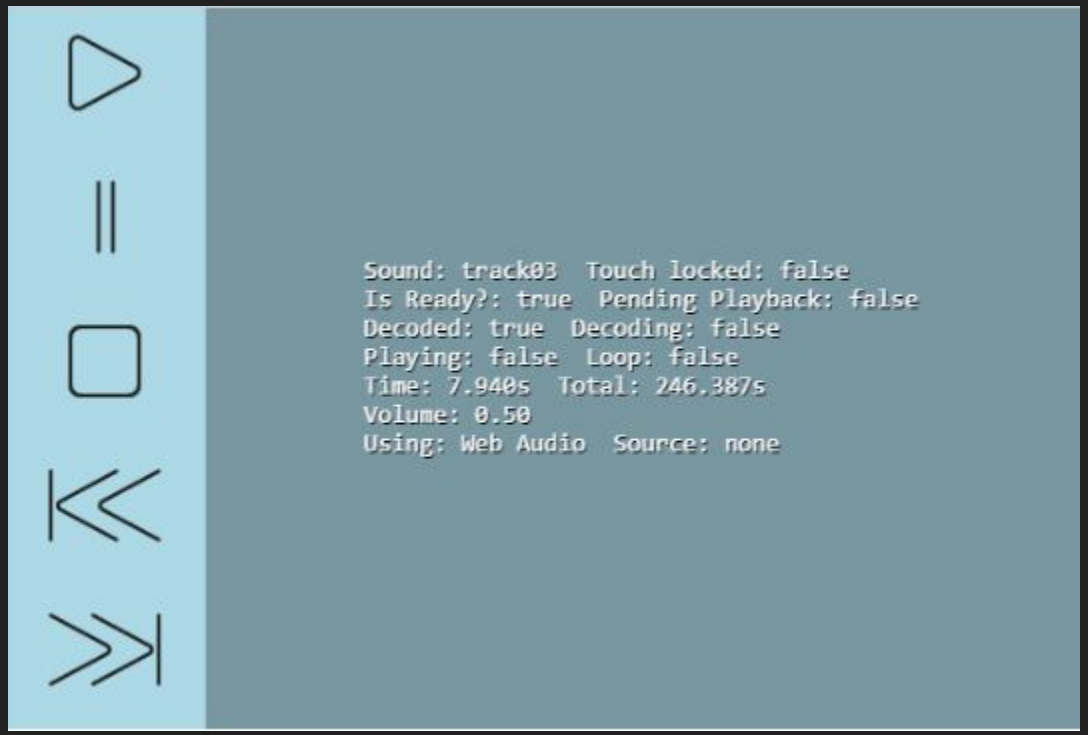
CMPM 120

Input, Movement, Physics, Collision

But first, audio

Audio



asset_management/assets05.js

Some parts of working with audio should be familiar:

```
// load audio
game.load.path = '../assets/music/';
game.load.audio('track01', ['dino.mp3']);
game.load.audio('track02', ['synthcart.mp3']);
game.load.audio('track03', ['sandserpents.mp3']);
```

Others are new:

```
update: function() {
    // wait for first mp3 to properly decode
    if(game.cache.isSoundDecoded('track01')) {
        game.state.start('Play');
    }
}
```

```
// add audio as an object hash table so we can do track management
this.tracks = {
    'track01' : game.add.audio('track01'),
    'track02' : game.add.audio('track02'),
    'track03' : game.add.audio('track03')
}
```

For our jukebox, we'll use a callback function:

```
this.play.input.useHandCursor = true;
this.play.events.onInputDown.add(this.playMusic, this);
```

Which interfaces with our `tracks` data structure and calls `play()`

```
playMusic: function() {
    let trackNum = 'track0' + this.currentTrack;
    // https://photonstorm.github.io/phaser-ce/Phaser.Sound.html
    // ('marker', start position, volume (0-1), loop)
    this.tracks[trackNum].play('', 0, 0.5, false);
},
```

Phaser.Input

Objectives

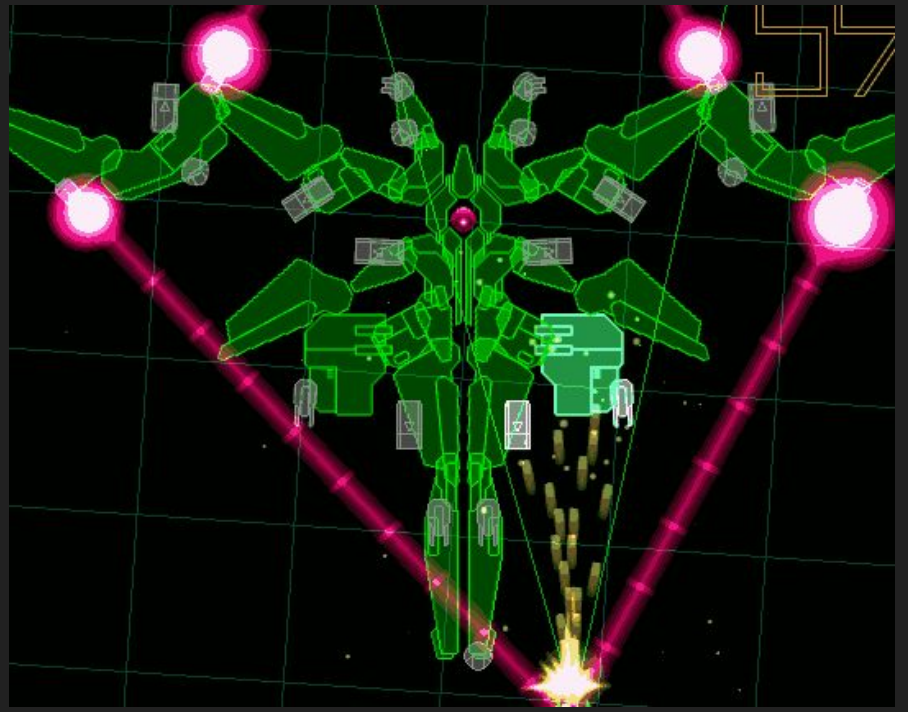
You will be able to...

- ➔ Describe how to read and process **input** for your games
- ➔ Discuss different kinds of **collisions** and **physics** systems
- ➔ Demonstrate **how to find and use** information about the **Phaser API** in the documentation

I'm making a vertical-scrolling bullet hell shooter with procedurally generated bosses

I want to use the keyboard to control the player's ship.

How do I do that?



Warning Forever

Phaser.Input

https://photonstorm.github.io/phaser-ce/Phaser.Input.html

Q: For my game, I want to get input from the WASD keys on the keyboard. How do I do it?

1. Go to the Phaser documentation
2. Individually, think about how to get the input
3. Briefly write down the line(s) of code that I need and where it should go

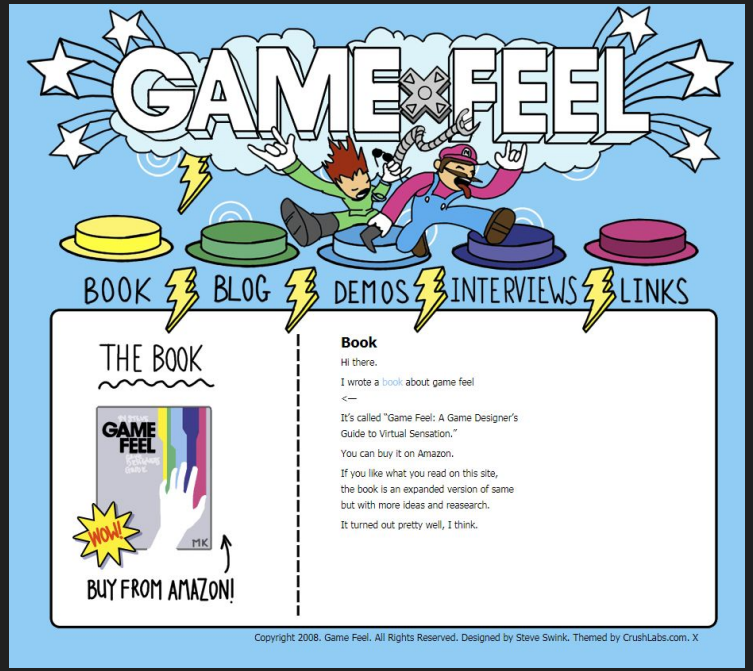
```
game.input.?
```

4. Pair with another student and discuss your solution

I also want to use the spacebar as my fire button, but when I press it, the browser page scrolls. What do I do?

Physics

How did jumping feel in your first assignment?



The library has a copy of *Game Feel*, by the way

Phaser's Physics Manager

`game.physics`

https://photonstorm.github.io/phaser-ce/#toc-19

Arcade Physics *or* P2 Physics

What is the difference?

Arcade vs P2

- Lightweight
- AABB-based collision
- Overlap
- Movement
- Velocity
- Acceleration
- Bullet pools

- Full-body advanced physics
- Multiple collision object shapes
- Overlap
- Movement
- Velocity
- Acceleration
- Contact materials
- Springs
- Constraints

A game object is limited to only one physics body
(and it can't be changed until the object is destroyed)

But you may have multiple systems active in a single game

acceleration allowGravity allowRotation angle angularAcceleration

angularDrag angularVelocity blocked bottom bounce center checkCollision

collideWorldBounds customSeparateX customSeparateY deltaMax dirty drag

embedded enable facing friction game gravity halfHeight halfWidth

height immovable isCircle isMoving left mass maxAngular maxVelocity

movementCallback movementCallbackContext moves newVelocity offset onCollide

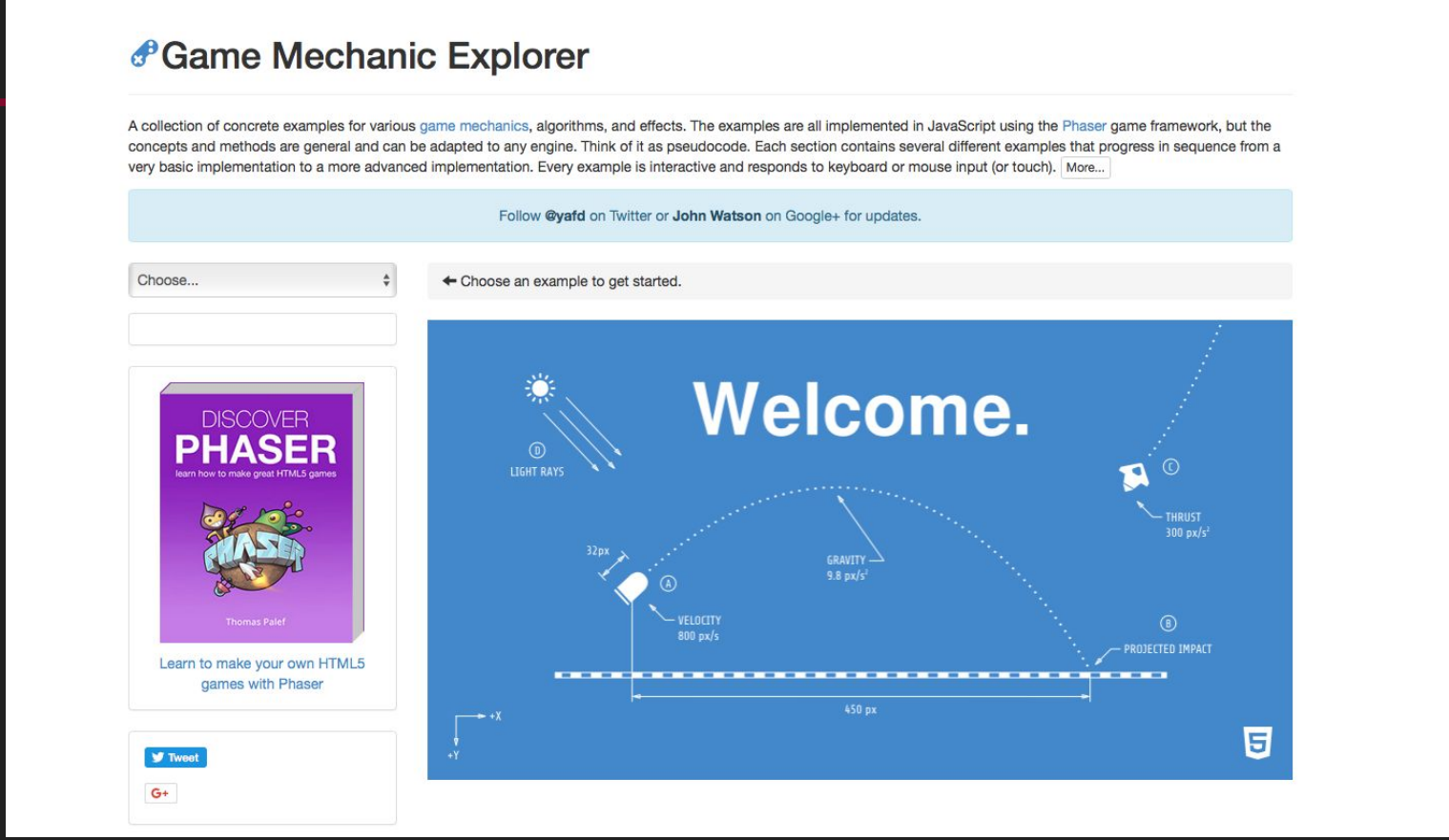
onMoveComplete onOverlap onWorldBounds overlapR overlapX overlapY position

preRotation prev radius right rotation skipQuadTree sourceHeight

sourceWidth speed sprite stopVelocityOnCollide syncBounds tilePadding top

touching type velocity wasTouching width worldBounce x y

Properties on an Arcade Physics body



https://gamemechanicexplorer.com/#

`Phaser.Sprite.anchor` sets the origin `Point` of the texture

(0,0)

default

(0.5,0.5)

```
sprite.anchor.set(0.5)
```

(1,1)

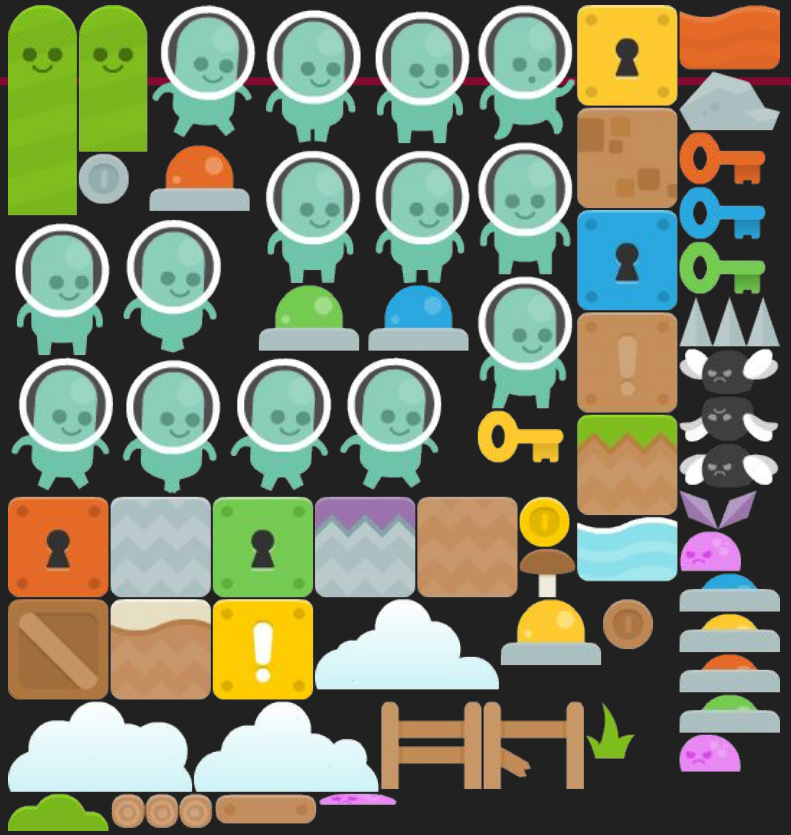
```
sprite.anchor.x = 1;
sprite.anchor.y = 1;
```

A Phaser `Point` object represents a location in a two-dimensional coordinate system, where x represents the horizontal axis and y represents the vertical axis.

```
// set up our alien sprite
this.alien = this.add.sprite(this.world.centerX, this.world.centerY, 'atlas', 'side');
this.alien.anchor.set(0.5);
this.alien.scale.setTo(this.HALFSCALE);
// set up alien physics
game.physics.enable(this.alien, Phaser.Physics.ARCADE);
```

https://www.leshylabs.com/apps/sstool/

```
2},"sourceSize":{"w":66,"h":92}},"jump"
4}},"side":{"frame":
:92}},"walk0001":{"frame":
6}},"walk0002":{"frame":
96}},"walk0003":{"frame":
96}},"walk0004":{"frame":
:96}},"walk0005":{"frame":
:96}},"walk0006":{"frame":
}},"walk0007":{"frame":
6}},"walk0008":{"frame":
96}},"walk0009":{"frame":
6}},"walk0010":{"frame":
96}},"walk0011":{"frame":
6}},"alien_plant":{"frame":
:28}},"block":{"frame":
70}},"bonus_used":{"frame":
:70}},"bonus":{"frame":
:70}},"bridge":{"frame":
24}},"bush":{"frame":
6}},"cloud_1":{"frame":
"h":63}},"cloud_2":{"frame":
```

The JSON specifies the frame locations in the texture atlas

https://photonstorm.github.io/phaser-ce/Phaser.Animation.html

```
this.alien = this.add.sprite(???);

this.alien.animations.add(???); // 1-frame idle animation
this.alien.animations.add(???); // 10-frame walk animation
```

Animation from an atlas

```
create: function() {
    // background color
    game.stage.backgroundColor = "#517B96";

    // wo ic it
    ga ics st r.
    ga ics gr = AR(
                    /IT
    // ou sp
    this.alien = this.add.sprite(this.world.centerX, this.world.centerY, 'atlas', 'side');
    this.alien.anchor.set(0.5);
    this.alien.scale.setTo(this.HALFSCALE);
    // set up alien physics
    game.physics.enable(this.alien, Phaser.Physics.ARCADE);
    this.alien.body.collideWorldBounds = true;
    // cap the alien's max velocity (x, y)
    this.alien.body.maxVelocity.setTo(this.MAX_VELOCITY, this.MAX_VELOCITY);
    // add drag to slow the physics body while not accelerating
    this.alien.body.drag.setTo(this.DRAG, 0);
    // set up alien animations
    // .add('key', [frames], frameRate, loop)
    // .generateFrameNames('prefix', start, stop, 'suffix', zeroPad) -> returns array
    // this handles atlas names in format: walk0001 - walk0011
    this.alien.animations.add('walk', Phaser.Animation.generateFrameNames('walk', 1, 11, '', 4), 30, true);
    this.alien.animations.add('idle', ['front'], 30, false);
```

Animation from an atlas

inputs01.js - inputs06.js

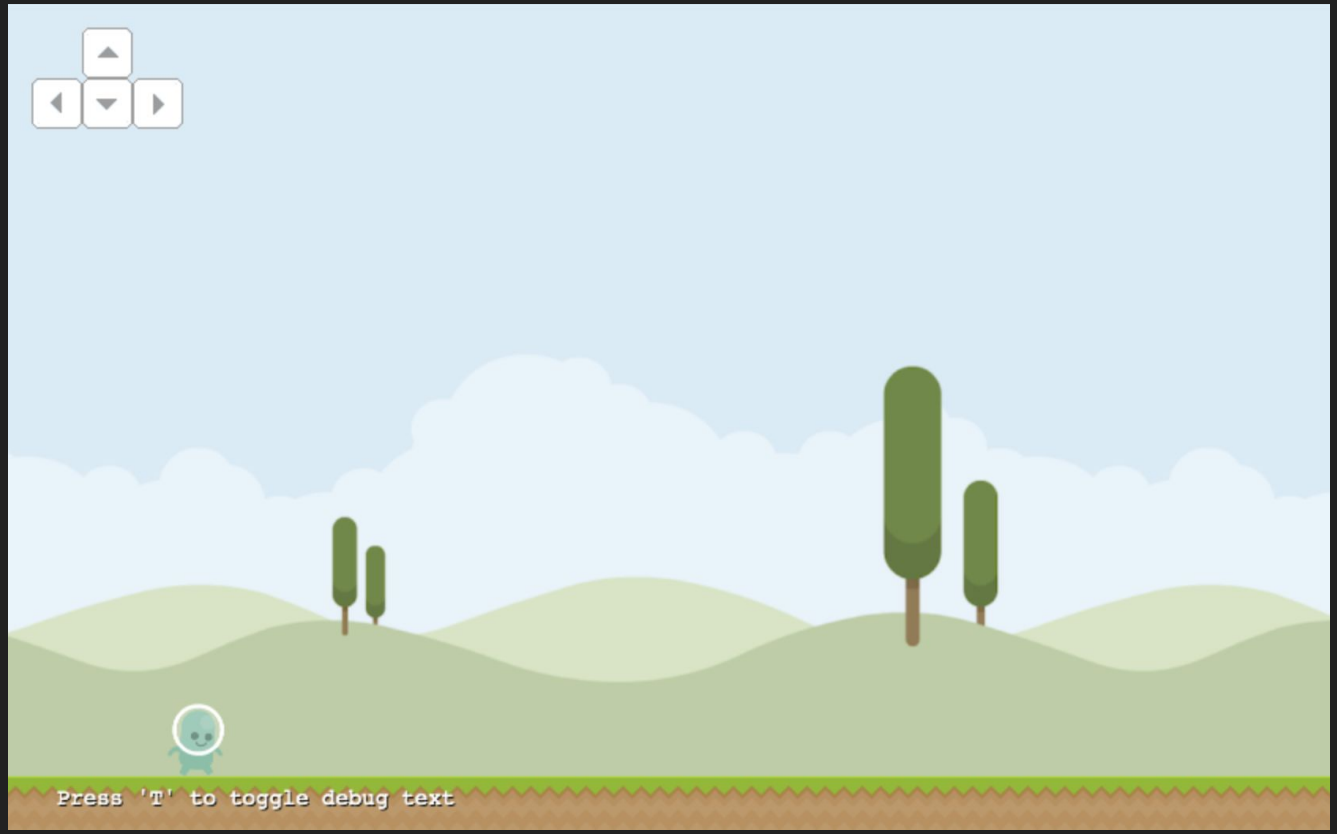
```
Inputs.Preloader.prototype = {
    preload: function() {
        // set load path and load texture atlas
        this.load.path = '../assets/img/';
        this.load.atlas('atlas', 'kenny_sheet.png', 'kenny_sheet.json');
        this.load.image('talltrees', 'talltrees.png');
    },
```

```
Inputs.Play.prototype = {
    preload: function() {

    },
    create: function() {
        // add bg as tile sprite
        this.talltrees = this.add.tileSprite(0,0, game.width, game.height, 'talltrees');

    update: function() {
        // update tileSprite background
        this.talltrees.tilePosition.x -= 8;
```

Tile Sprites


Press 'T' to toggle debug text

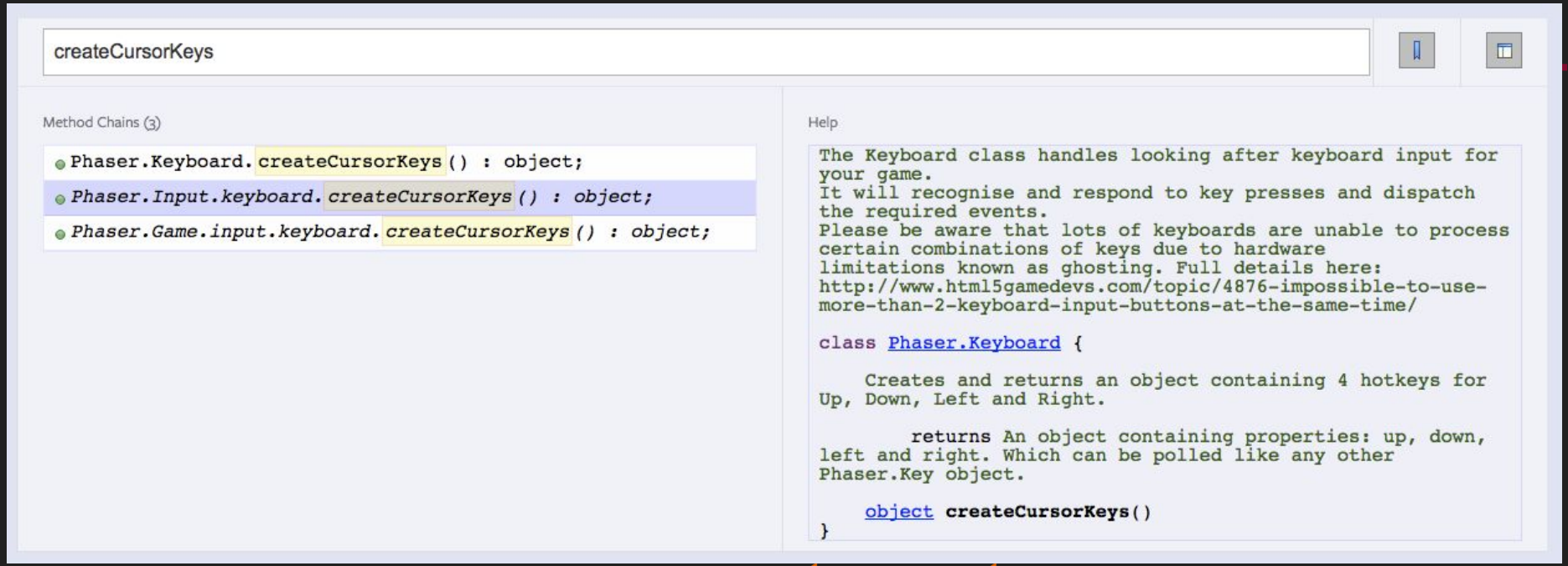
```
create: function() {
    // start up arcade physics
    this.physics.startSystem(Phaser.Physics.ARCADE);

    // add our alien sprite and set anchor so sprite flipping looks correct
    this.alien = this.add.sprite(this.world.centerX, this.world.centerY, 'atlas', 'front');
    this.alien.anchor.set(0.5);

    // apply physics to alien...
    this.physics.enable(this.alien, Phaser.Physics.ARCADE);
    // ...and adjust settings
    this.alien.body.drag.set(100);          // drag applied to motion
    this.alien.body.maxVelocity.set(300);   // max velocity that body can reach in px/sec^2

    // setup input
    cursors = this.input.keyboard.createCursorKeys();
    //this.input.keyboard.addKeyCapture([Phaser.Keyboard.SPACEBAR]);
},
update: function() {
    // check keyboard input and move alien appropriately
    if(cursors.up.isDown) {
        // accelerationFromRotation(rotation in radians, speed in px/sec^2, point x/y
        acceleration)
        this.physics.arcade.accelerationFromRotation(this.alien.rotation-Math.PI/2, 200, this.
            alien.body.acceleration);
    } else {
        this.alien.body.acceleration.set(0);
    }

    if(cursors.left.isDown) {
        this.alien.body.angularVelocity = -300; // rate of change of angular pos of rotating body
    } else if (cursors.right.isDown) {
        this.alien.body.angularVelocity = 300;
    } else {
        this.alien.body.angularVelocity = 0;
    }
```



"Creates and returns an object containing 4 🔥hotkeys🔥 for Up, Down, Left, and Right."

Returns an object (`cursors`) containing four properties (`cursors.up`, `cursors.down`, `cursors.left`, `cursors.right`) that can be polled for keyboard input.

What if you need individual key presses?

One Strategy

create()

```
    // create a new Phaser Key object for a single key
    this.jumpKey = game.input.keyboard.addKey(Phaser.Keyboard.SPACEBAR);
```

update()

```
    // did the jump key happen this frame?
    if(this.jumpKey.justPressed()) {
        console.log('kick!');
    }
```

Another Strategy

create()

```
    this.hugKey = game.input.keyboard.addKey(Phaser.Keyboard.H);
    this.hugKey.onDown.add(this.doHug, this);
```

A callback function

```
doHug: function() {
    console.log('Hugging you :)');
},
```

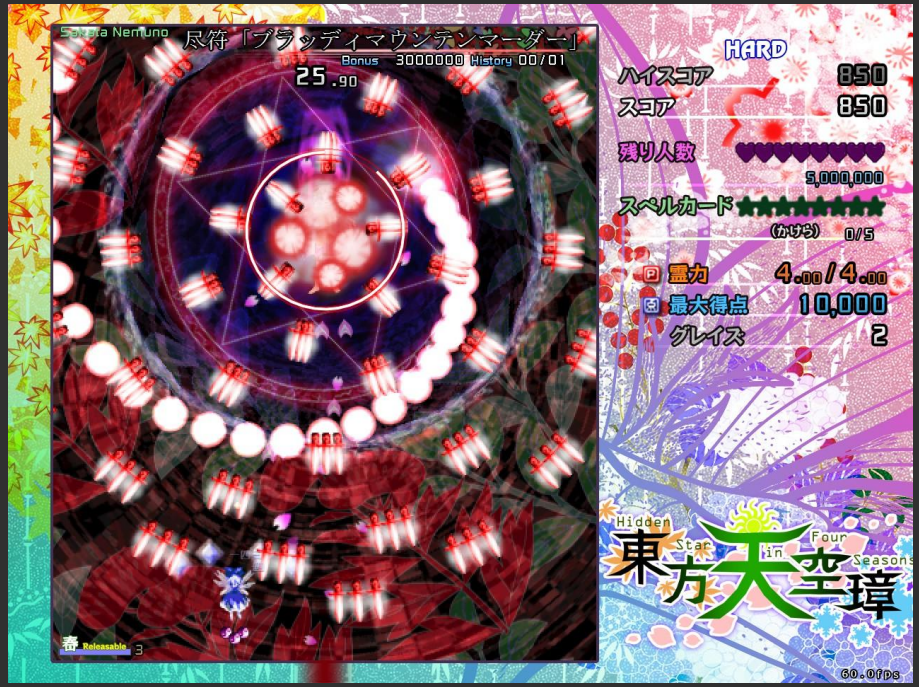
inputs07.js

A touching look at

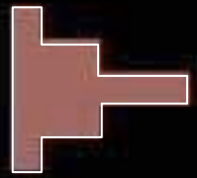
Collision Detection

There are multiple collision processes

- **Collision detection** determines **if** two objects are coincident.
- **Collision determination** asks **when** objects came into contact and where they intersect.
- **Collision resolution** defines **how** the program should respond to the collision.

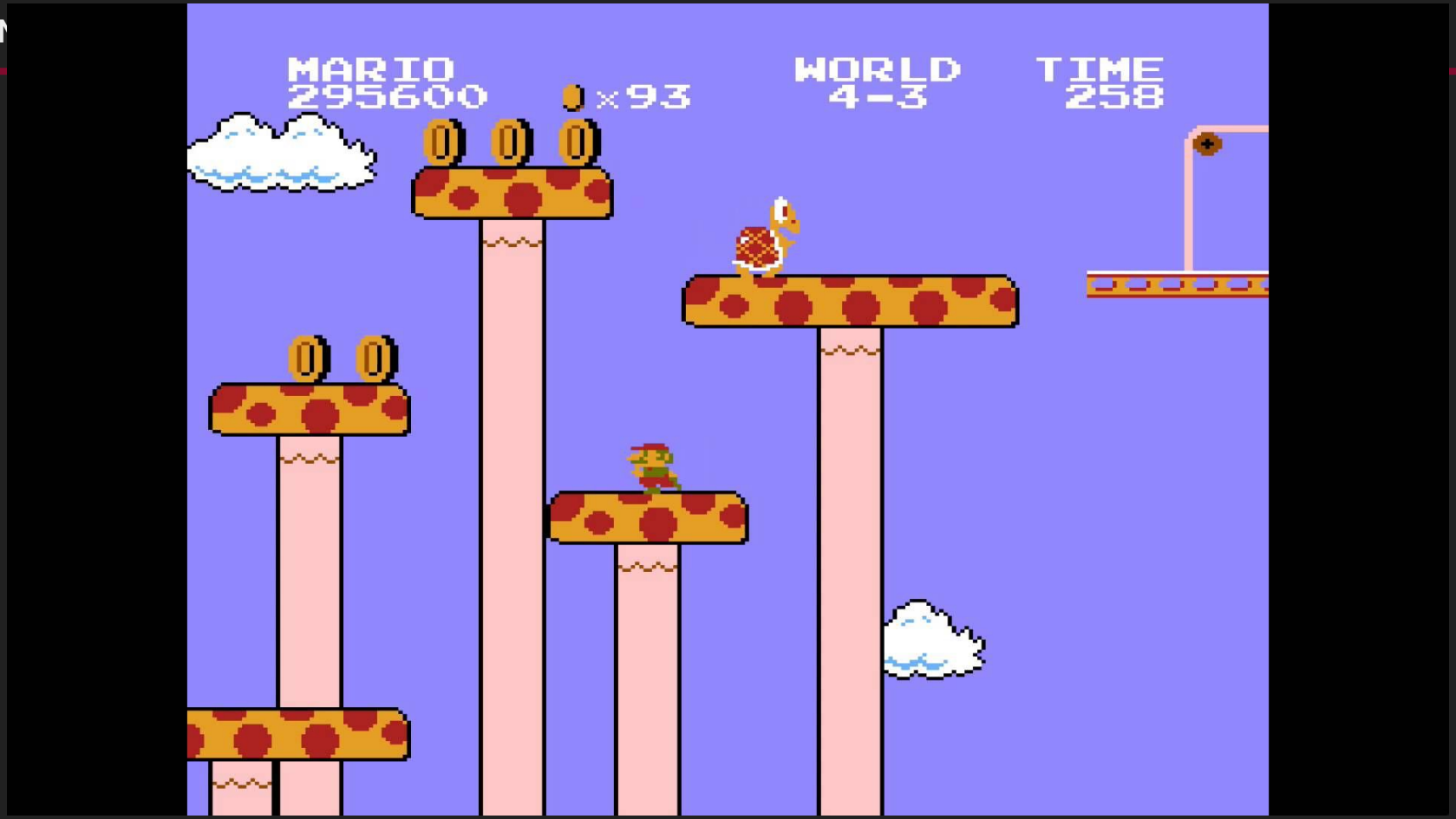



4270
2



Q: How was this possible in the 1970s?


A: Not a lot going on
86 12



Hardware/Software Collision Detection

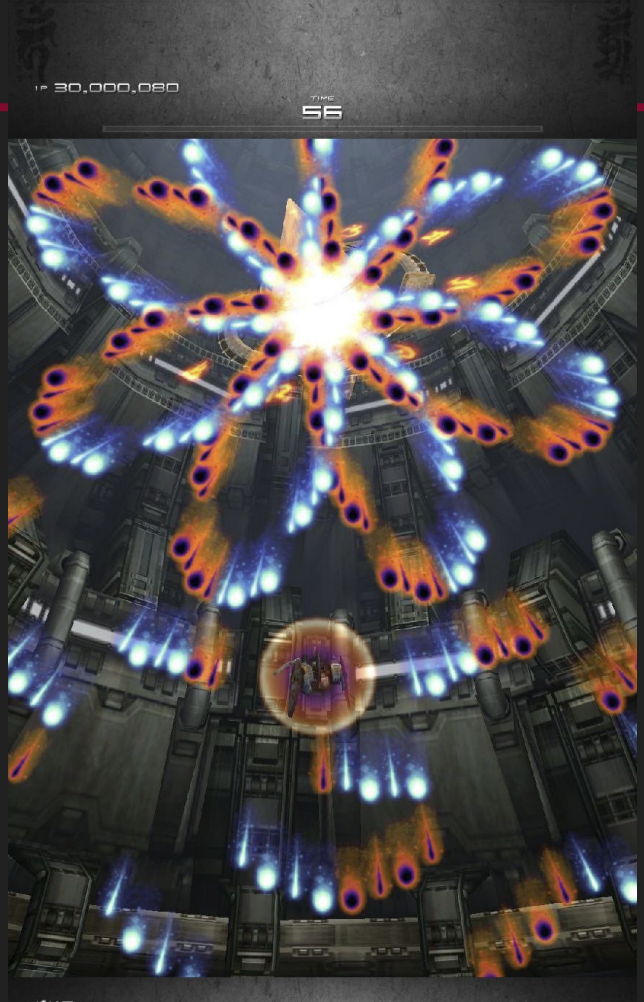
How does collision detection scale?

Deathsmiles (2007)

Naive collision detection assumes that all n objects onscreen may potentially intersect with one another, so it checks every object against every other object.

That's **n(n-1)/2** checks per frame.

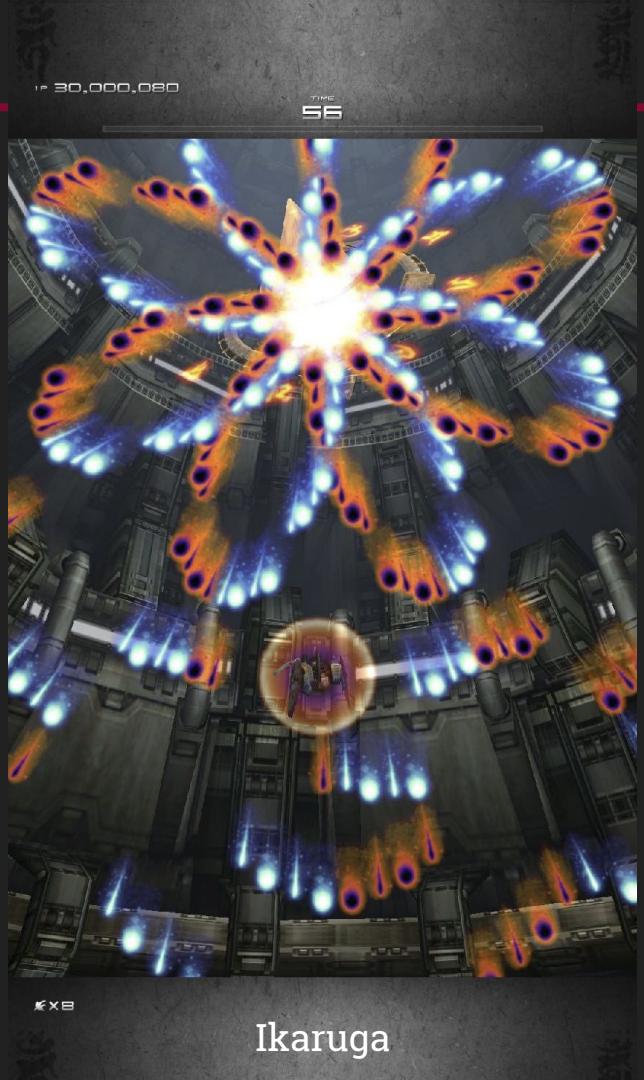
```
n=3    ≡ 3 checks
n=8    ≡ 28 checks
n=64   ≡ 2016 checks
n=128  ≡ 8128 checks
n=500  ≡ 124,750 checks
n=5000 ≡ 12,497,500 checks
```



Ikaruga

What are the problems this causes?

- ➔ Different behavior
- ➔ ?
- ➔ ?



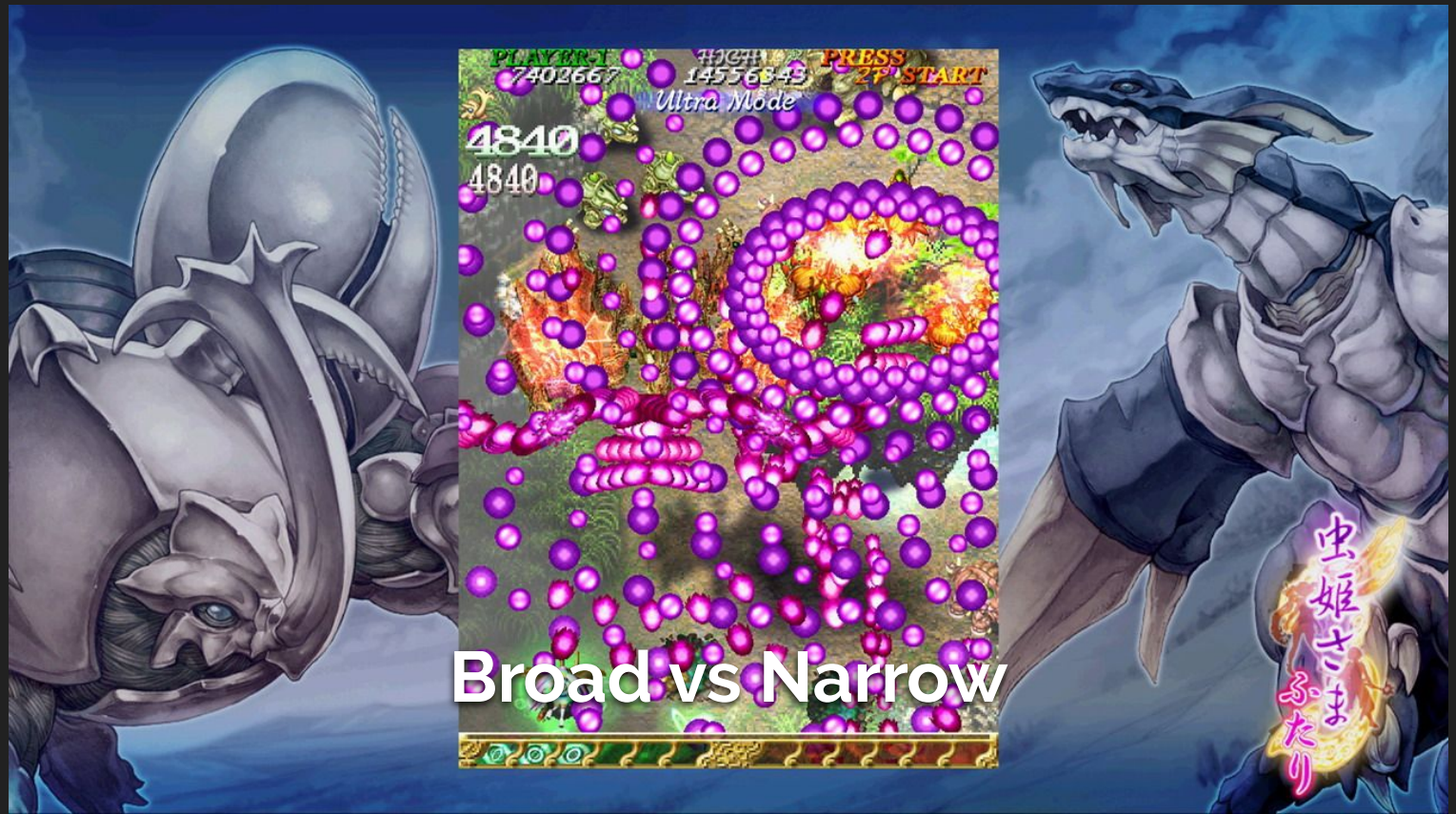
Ikaruga

How can we address this problem?

- ➔ ?
- ➔ ?

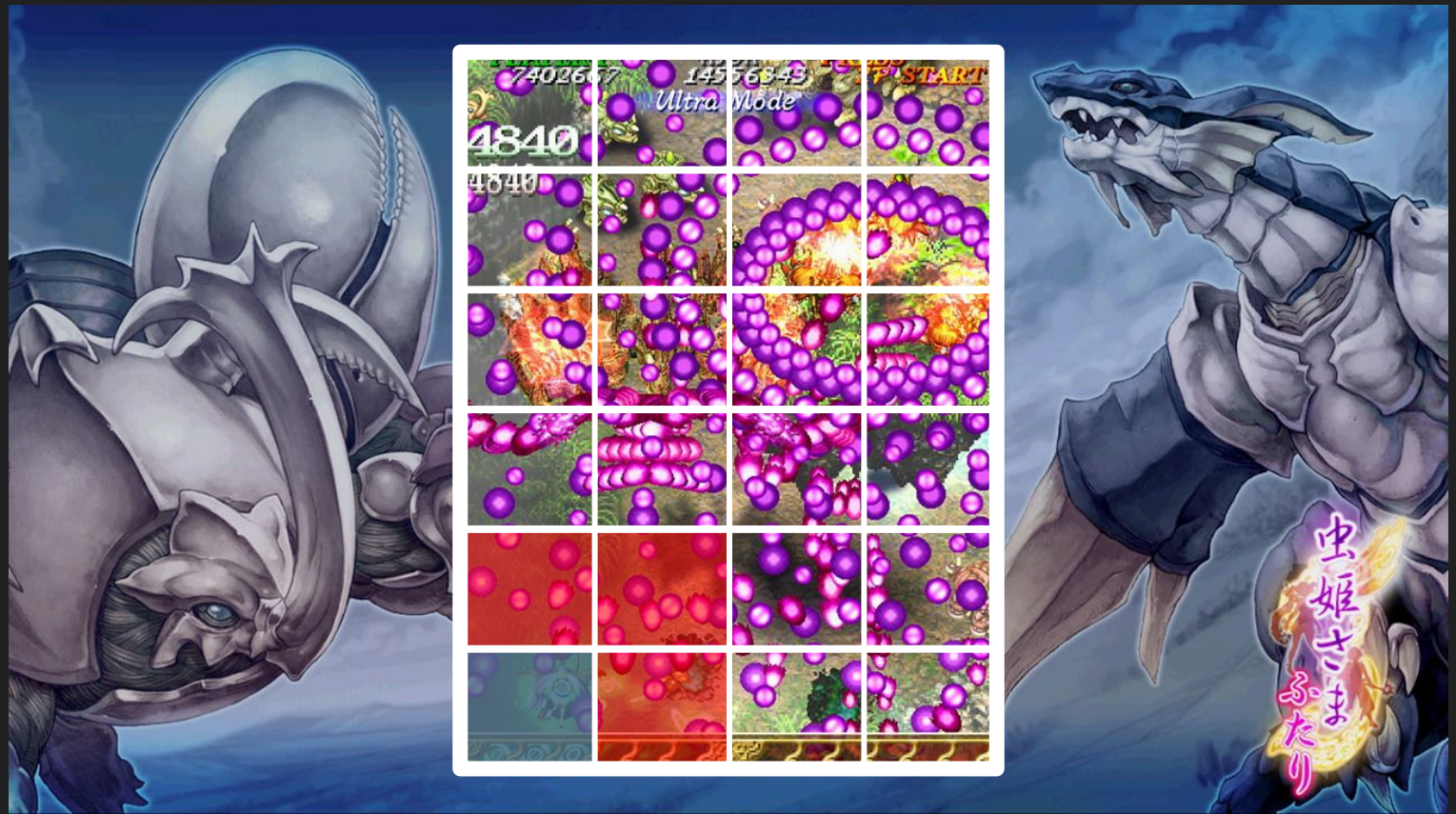
Ikaruga

Reducing **number** of collision pairs



Broad vs Narrow

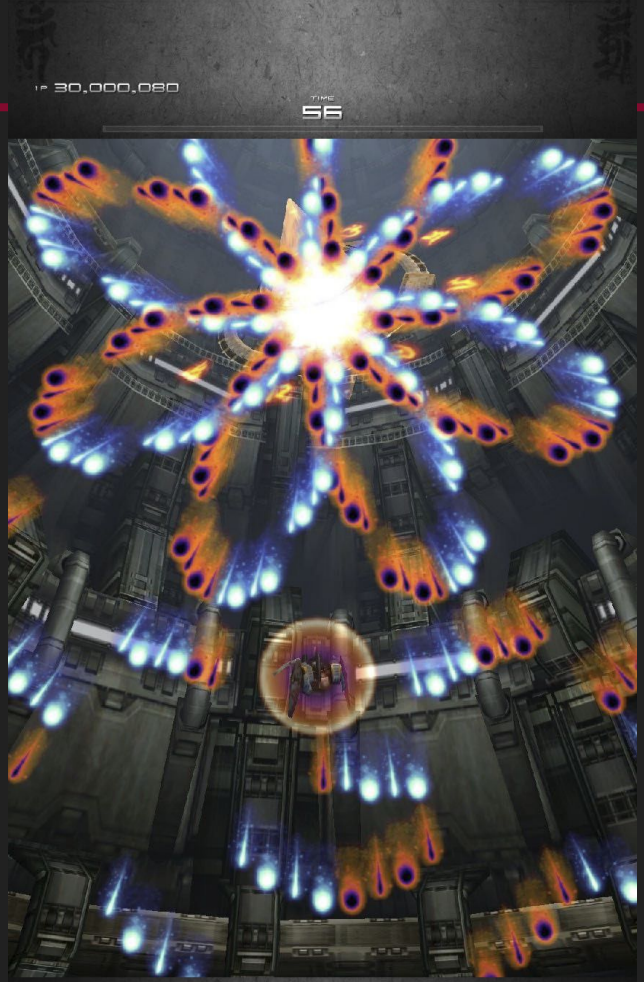
Broadphase Sweep



Narrowphase Sweep

Genre matters!

In a bullet-hell shooter, collisions with the player's ship are important, but enemy bullets might pass through every other object onscreen.



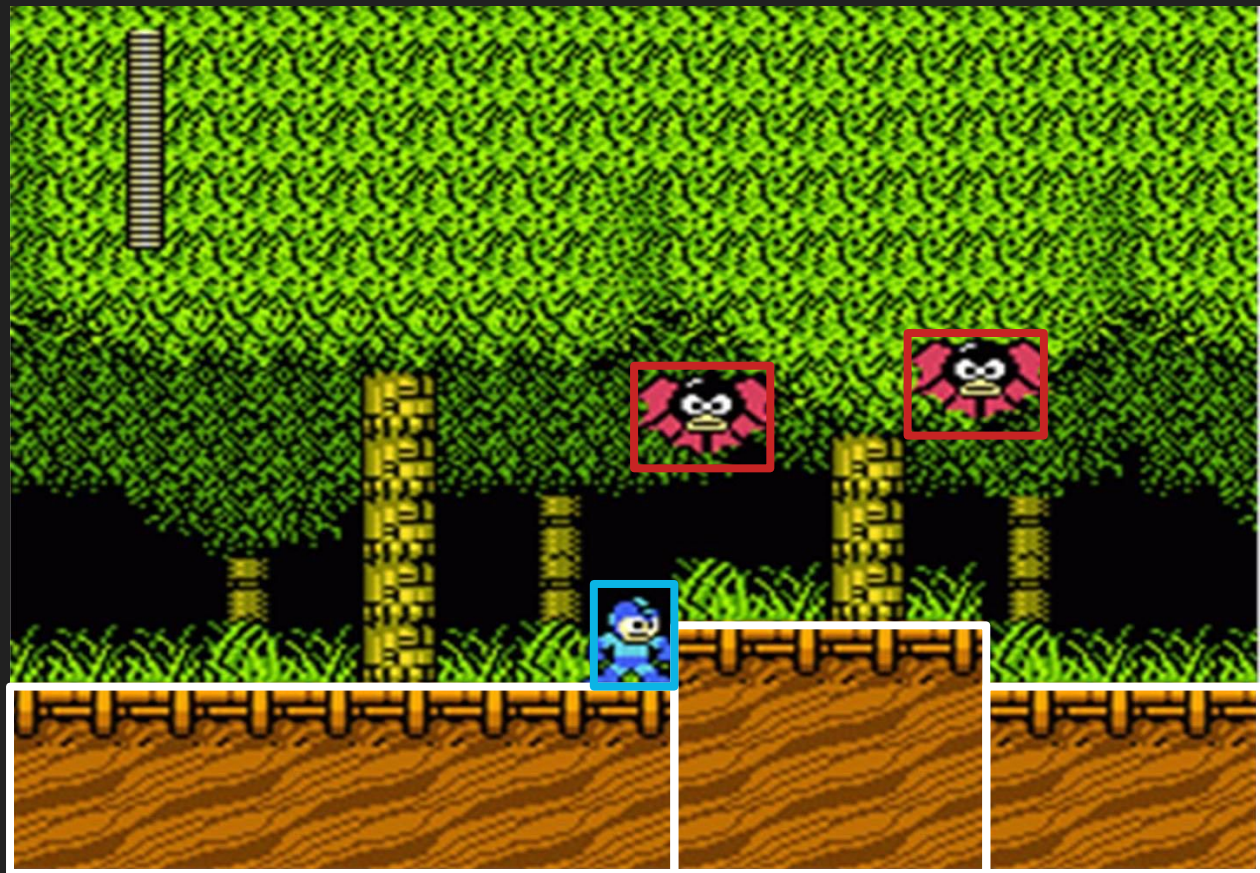
Ikaruga

Reducing the cost of collision checks

Bounding volumes

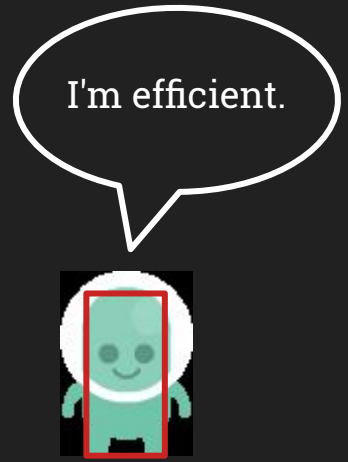
Reducing the cost of collision checks

Bounding volumes

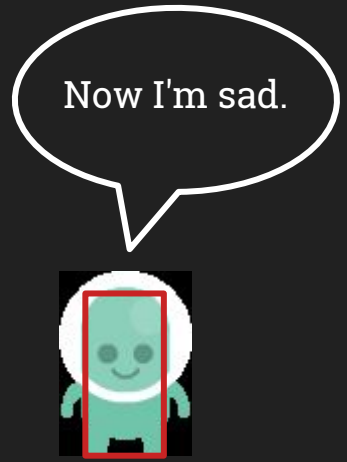


Advantages of Bounding Volumes

- Inexpensive intersection tests
- Reduced complexity of objects
- Inexpensive to compute
- Easy to rotate and transform
- Low memory use



Disadvantages of Bounding Volumes?



Common Bounding Volumes

circle/sphere

axis-aligned bounding box

oriented bounding box

convex hull

AXIS-ALIGNED BOUNDING BOX (AABB)

Checks collisions between two axis-aligned (i.e. not rotated) rectangles by ensuring that there are no gaps between any of the four sides of the rectangles.

```
1
2  // simple AABB collision check
3
4  var rect1 = { x: 10, y: 10, w: 50, h: 50 };
5
6  var rect2 = { x: 50, y:50, w: 210, h: 30 };
7
8  var rect3 = { x: 250, y: 20, w: 100, h: 180 };
9
10 var rect4 = { x: 140, y: 70, w: 20, h: 280 };
11
12 function checkAABB(rectA, rectB) {
13     if( rectA.x < rectB.x + rectB.w &&
14         rectA.x + rectA.w > rectB.x &&
15         rectA.y < rectB.y + rectB.h &&
16         rectA.h + rectA.y > rectB.y) {
17         return true;
18     } else {
19         return false;
20     }
21 }
22
23 checkAABB(rect1, rect2); // true
24 checkAABB(rect2, rect3); // true
25 checkAABB(rect2, rect4); // true
26 checkAABB(rect1, rect3); // false
27 checkAABB(rect3, rect4); // false
```

Phaser Arcade Physics

Arcade uses AABB collision only, so it's **fast** and **cheap** to compute (but less **precise**).

Arcade physics checks collisions for:
sprite v. sprite, sprite v. group, group v. group.

Collisions will separate the objects that collided, pushing them back to a state where they do not overlap visually.

Collision checks need to be done in the **update()** function.

Collision handlers are callback functions that respond to an overlap between two display objects.

All collision handlers in Phaser take two arguments, which will be references to the objects that hit each other.

```
// Phaser collision - called in update()
game.physics.arcade.collide(objectOne, objectTwo);
```

To use collision, you must first **enable** physics bodies on a sprite.

Once enabled, it's possible to access all of the properties and methods of a sprite's body, such as acceleration, velocity, bounce, etc.

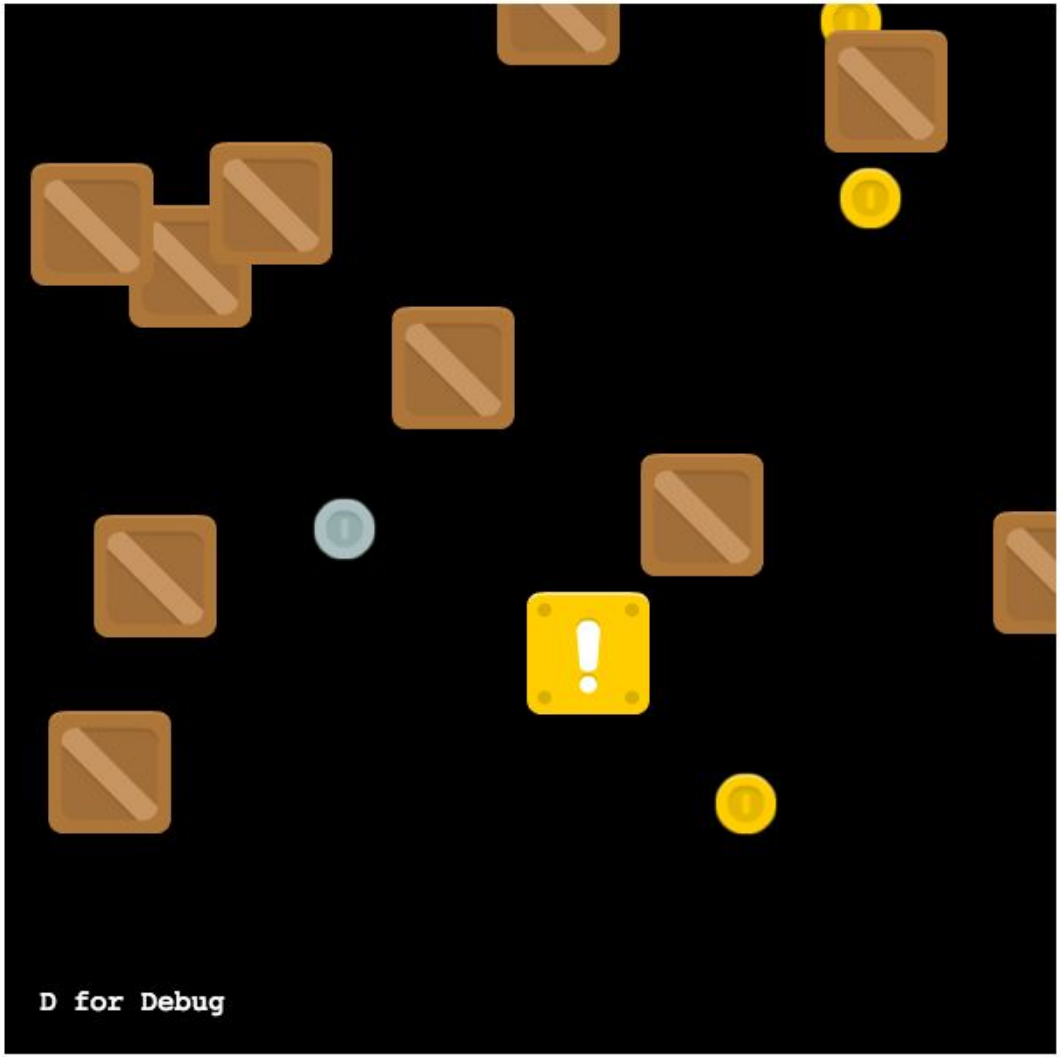
```
// setup player physics - called in create()
game.physics.enable(game.player);              // enable physics (default ARCADE)
game.player.body.drag.set(50);                 // set fricion
game.player.body.maxVelocity.set(250);         // set max velocity
game.player.body.collideWorldBounds = true; // player limited to world edges
game.player.body.bounce.set(0.25);             // player is bouncy
```

If you don't want physics bodies to automatically separate, you may use an **overlap** call instead.

There are also optional overlap and process callback functions that allow you to respond to the overlap in different ways.

```
// overlap with callback
game.physics.arcade.overlap(object1, group1, overlapCallback, processCallback, this);

// the two objects are passed to the callback in the order you specified
overlapCallback: function(obj1, g1) {
    // respond to overlap
},
processCallback: function(obj1, g1) {
    // perform additional check on overlap before handing off to overlapCallback
}
```


D for Debug

But last, your Endless Runner

Organization (2.5 points)

Comments	Logically comment your source to demonstrate that you understand how each section works. (0.5)
Organization	Your file structure is organized logically and legibly (0.5)
No Errors	Game runs from localhost (0.5) with no code errors (0.5).
Submit	Submit your project to Canvas as a .zip that includes the framework so the graders can run it. (0.5)

Structure and Design I

3 States	Have at least three states: a main menu (0.5), a state where you play the actual game (0.5), and a game over state (0.5). You may name these however you like. You may also have more, depending on how you structure your game.	**Collision Detection**	Properly use collision detection (0.5).
Instructions	Communicate how to play w/ clear instructions (0.5).	**Background Music**	Have looping background music (0.5).
State Transitions	Properly transition between states and allow the player to restart w/out having to reload the page (0.5).	**Sound Effects**	Use sound effects for key mechanics and/or events (0.5) according to your design.
Player Input	Have some form of player input/control (0.5) according to your design.	**Randomness**	Use randomness to generate challenge, e.g. terrain, pickups, etc. (0.5).

Structure and Design II

Animated Character	Include an animated character(s) (0.5) that use a texture atlas (0.5).	**Metric**	Include some metric of accomplishment that a player can improve over time, e.g., score (0.5).
Simulate Scrolling	Simulate scrolling, e.g., tilesprite (0.5).	**Endless**	Be theoretically endless (0.5).
Playable	Be playable for at least 15 seconds for a new player of low to moderate skill (0.5).		

Creative Tilt (2 points)

Does your game...

Technical Interest	...do something technically interesting? Are you particularly proud of a programming technique you implemented? Did you look beyond the class examples and learn how to do something new? (1)
Visual Style	...have a great visual style? Does it use music or art that you created? Are you trying something new or clever with the endless runner form? (1)