Input, Movement, Physics, Collision
But first, audio
CMPM 120

Audio

asset_management/assets05.js
Some parts of working with audio should be familiar:

```javascript
// 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:

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

```javascript
// 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:

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

Which interfaces with our **tracks** data structure and calls `play()`
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?
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
Properties on an Arcade Physics body
Game Mechanic Explorer

A collection of concrete examples for various game mechanics, algorithms, and effects. The examples are all implemented in JavaScript using the Phaser game framework, but the concepts and methods are general and can be adapted to any engine. Think of it as pseudocode. Each section contains several different examples that progress in sequence from a very basic implementation to a more advanced implementation. Every example is interactive and responds to keyboard or mouse input (or touch). More...

Follow @yaff on Twitter or John Watson on Google+ for updates.

Choose...

Choose an example to get started.

Welcome.

https://gamemechanicexplorer.com/
**Phaser.Sprite.anchor** sets the origin **Point** of the texture

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.

```javascript
sprite.anchor.set(0.5); // (0.5, 0.5)
sprite.anchor.x = 1; // (1, 1)
sprite.anchor.y = 1;
```
```javascript
// 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/
The JSON specifies the frame locations in the texture atlas
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";
    // world gravity and physics
    game.physics.enable(this.world, Phaser.Physics.ARCADE);
    game.physics.arcade.gravity.y = 800;
    // set up player physics
    player.body.maxVelocity.setTo(this.player.MAX_VELOCITY, this.player.MAX_VELOCITY);
    player.body.drag.setTo(this.player.DRAG, 0);
    // set up player animations
    // .add('key', [frames], frameRate, loop)
    // .generateFrameNames('prefix', start, stop, 'suffix', zeroPad) -> returns array
    // this handles atlas names in format: walk0001 - walk0011
    this.player.animations.add('run', Phaser.Animation.generateFrameNames('run', 1, 11, '', 4), 30, true);
    this.player.animations.add('run', Phaser.Animation.generateFrameNames('run', 1, 11, '', 4), 30, false);
}
inputs01.js - inputs06.js
Tile Sprites

```javascript
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');
    },
    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;
    }

Inputs.Play.prototype = {
    preload: function() {
    },
    create: function() {
    },
    update: function() {
    }
};
```
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();
}
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()

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

update()

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

Another Strategy

create()

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

A callback function

```javascript
doHug: function() {
    console.log('Hugging you :(');
},
```
Press 'T' to toggle debug text
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.
Q: How was this possible in the 1970s?
A: Not a lot going on
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.

- $n=3 \implies 3$ checks
- $n=8 \implies 28$ checks
- $n=64 \implies 2016$ checks
- $n=128 \implies 8128$ checks
- $n=500 \implies 124,750$ checks
- $n=5000 \implies 12,497,500$ checks
What are the problems this causes?

➔ Different behavior

➔ ?

➔ ?
How can we address this problem?

➔ ?

➔ ?
Reducing **number** of collision pairs
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.
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?

Now I'm sad.
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.
// simple AABB collision check

var rect1 = { x: 10, y: 10, w: 50, h: 50 };
var rect2 = { x: 50, y: 50, w: 210, h: 30 };
var rect3 = { x: 250, y: 20, w: 100, h: 180 };
var rect4 = { x: 140, y: 70, w: 20, h: 280 };

function checkAABB(rectA, rectB) {
    if (rectA.x < rectB.x + rectB.w &&
        rectA.x + rectA.w > rectB.x &&
        rectA.y < rectB.y + rectB.h &&
        rectA.h + rectA.y > rectB.y) {
        return true;
    } else {
        return false;
    }
}

checkAABB(rect1, rect2); // true
checkAABB(rect2, rect3); // true
checkAABB(rect2, rect4); // true
checkAABB(rect2, rect4); // false
checkAABB(rect1, rect3); // false
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.

```javascript
// 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.

```javascript
// setup player physics - called in create()
game.physics.enable(game.player); // enable physics (default ARCADE)
game.player.body.drag.set(50); // set friccion
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.

```javascript
// 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)

<table>
<thead>
<tr>
<th><strong>Comments</strong></th>
<th>Logically comment your source to demonstrate that you understand how each section works. (0.5)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Organization</strong></td>
<td>Your file structure is organized logically and legibly (0.5)</td>
</tr>
<tr>
<td><strong>No Errors</strong></td>
<td>Game runs from localhost (0.5) with no code errors (0.5).</td>
</tr>
<tr>
<td><strong>Submit</strong></td>
<td>Submit your project to Canvas as a .zip that includes the framework so the graders can run it. (0.5)</td>
</tr>
</tbody>
</table>
# Structure and Design I

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>3 States</strong></td>
<td>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.</td>
<td>1.5</td>
</tr>
<tr>
<td><strong>Collision Detection</strong></td>
<td>Properly use collision detection (0.5).</td>
<td></td>
</tr>
<tr>
<td><strong>Instructions</strong></td>
<td>Communicate how to play w/ clear instructions (0.5).</td>
<td>0.5</td>
</tr>
<tr>
<td><strong>Background Music</strong></td>
<td>Have looping background music (0.5).</td>
<td>0.5</td>
</tr>
<tr>
<td><strong>State Transitions</strong></td>
<td>Properly transition between states and allow the player to restart w/out having to reload the page (0.5).</td>
<td>0.5</td>
</tr>
<tr>
<td><strong>Sound Effects</strong></td>
<td>Use sound effects for key mechanics and/or events (0.5) according to your design.</td>
<td>0.5</td>
</tr>
<tr>
<td><strong>Player Input</strong></td>
<td>Have some form of player input/control (0.5) according to your design.</td>
<td>0.5</td>
</tr>
<tr>
<td><strong>Randomness</strong></td>
<td>Use randomness to generate challenge, e.g. terrain, pickups, etc. (0.5).</td>
<td>0.5</td>
</tr>
</tbody>
</table>
## Structure and Design II

<table>
<thead>
<tr>
<th>Animated Character</th>
<th>Include an animated character(s) (0.5) that use a texture atlas (0.5).</th>
<th>Metric</th>
<th>Include some metric of accomplishment that a player can improve over time, e.g., score (0.5).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulate Scrolling</td>
<td>Simulate scrolling, e.g., tilesprite (0.5).</td>
<td>Endless</td>
<td>Be theoretically endless (0.5).</td>
</tr>
<tr>
<td>Playable</td>
<td>Be playable for at least 15 seconds for a new player of low to moderate skill (0.5).</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Creative Tilt (2 points)

Does your game...

<table>
<thead>
<tr>
<th>Technical Interest</th>
<th>...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)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visual Style</td>
<td>...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)</td>
</tr>
</tbody>
</table>