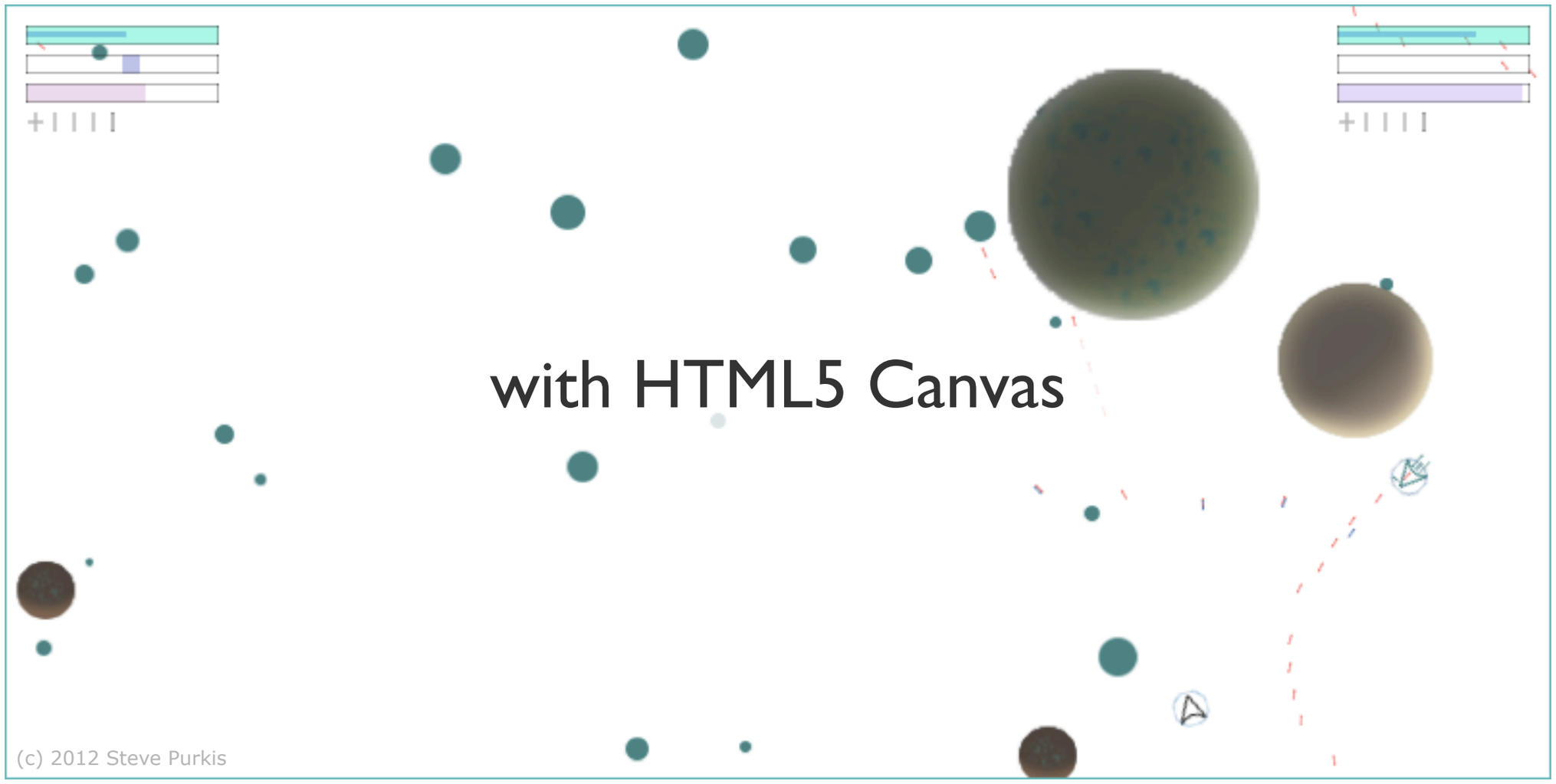
Asteroid(s)*



* Asteroids™ is a trademark of Atari Inc.

About Steve

Software Dev + Manager

not really a front-end dev

nor a game dev

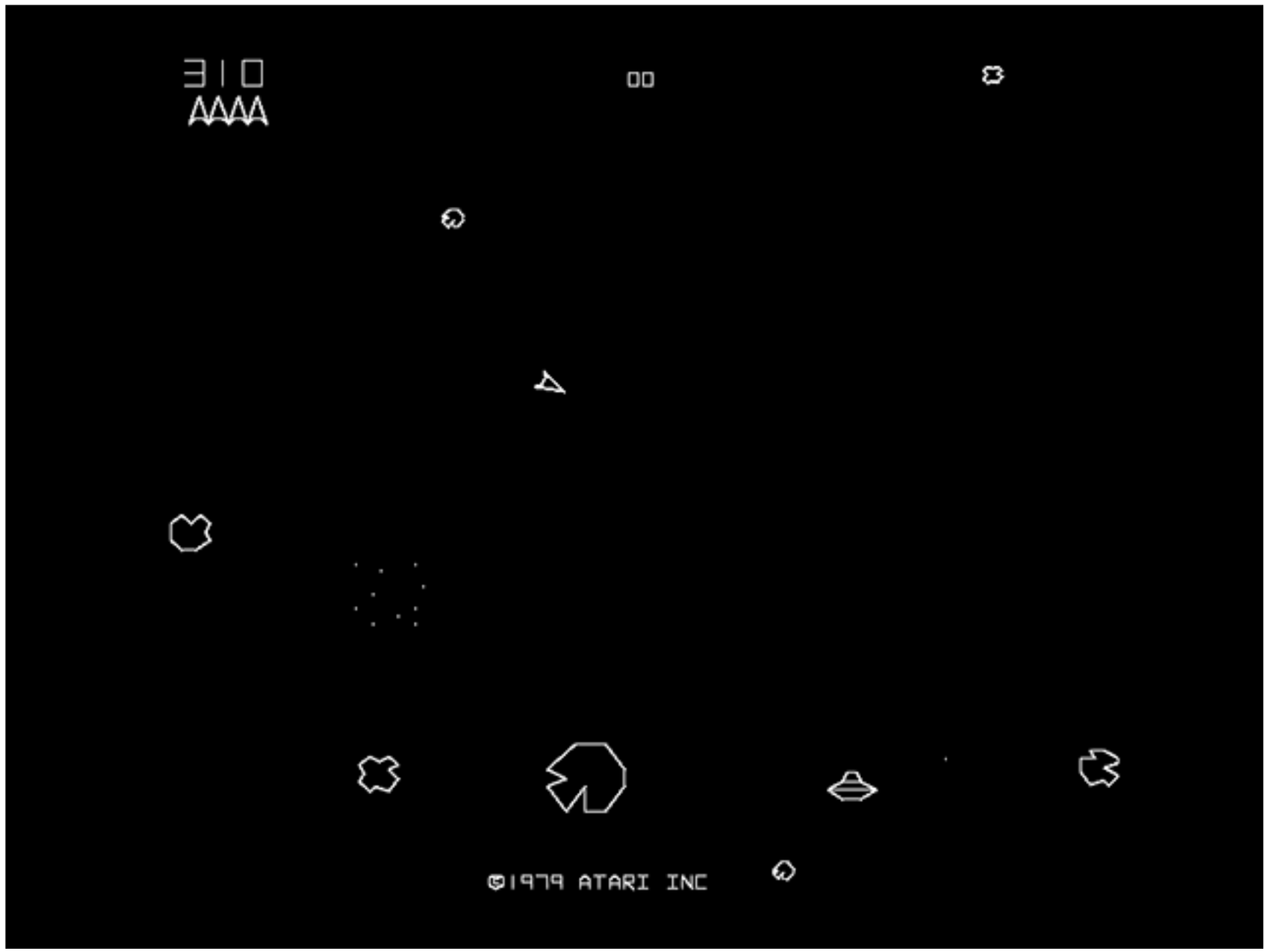
(but I like to play!)

Uhh...
Asteroids?

What's that, then?

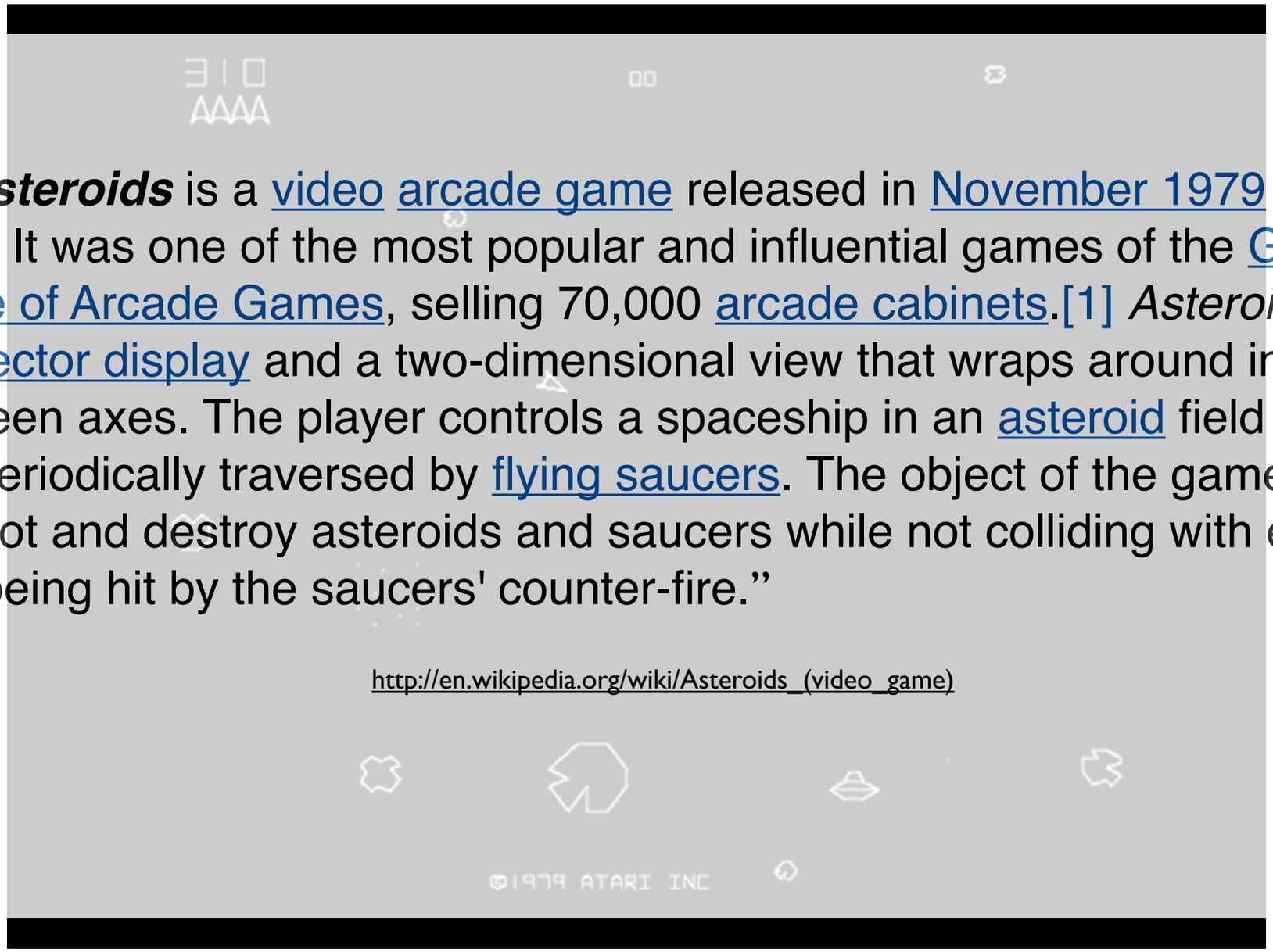
http://www.flickr.com/photos/onkel_wart/201905222/

Asteroids™



http://en.wikipedia.org/wiki/File:Asteroi1.png

Asteroids™



“ ***Asteroids*** is a video arcade game released in November 1979 by Atari Inc. It was one of the most popular and influential games of the Golden Age of Arcade Games, selling 70,000 arcade cabinets.[1] *Asteroids* uses a vector display and a two-dimensional view that wraps around in both screen axes. The player controls a spaceship in an asteroid field which is periodically traversed by flying saucers. The object of the game is to shoot and destroy asteroids and saucers while not colliding with either, or being hit by the saucers' counter-fire.”

http://en.wikipedia.org/wiki/Asteroids_(video_game)

http://en.wikipedia.org/wiki/File:Asteroi1.png

Asteroids™

Note that the term “Asteroids” is © Atari when used with a game.
I didn’t know that when I wrote this...
Oops.

Atari:

- I promise I’ll change the name of the game.
- In the meantime, consider this free marketing! :-)

Why DIY Asteroid(s)?

- Yes, it's been done before.
- Yes, I could have used a Game Engine.
- Yes, I could have used open|web GL.
- No, I'm not a "Not Invented Here" guy.

Why not?

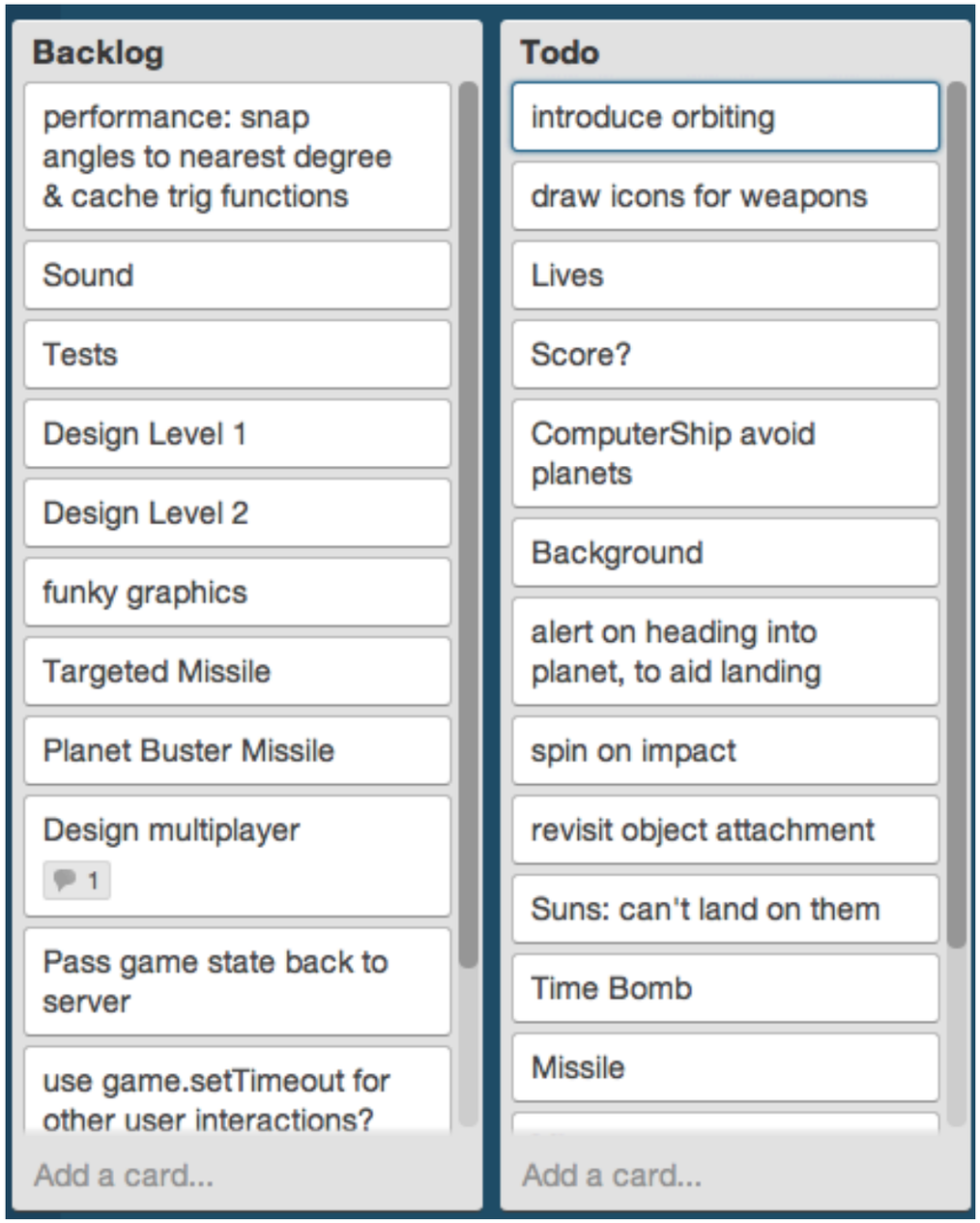
- It's a fun way to learn new tech.
- I learn better by doing.

(ok, so I've secretly wanted to write my own version of asteroids since I was a kid.)

I was learning about HTML5 (yes I know it came out several years ago. I've been busy).
A few years ago, the only way you'd be able to do this is with Flash.

demo

It's not Done...

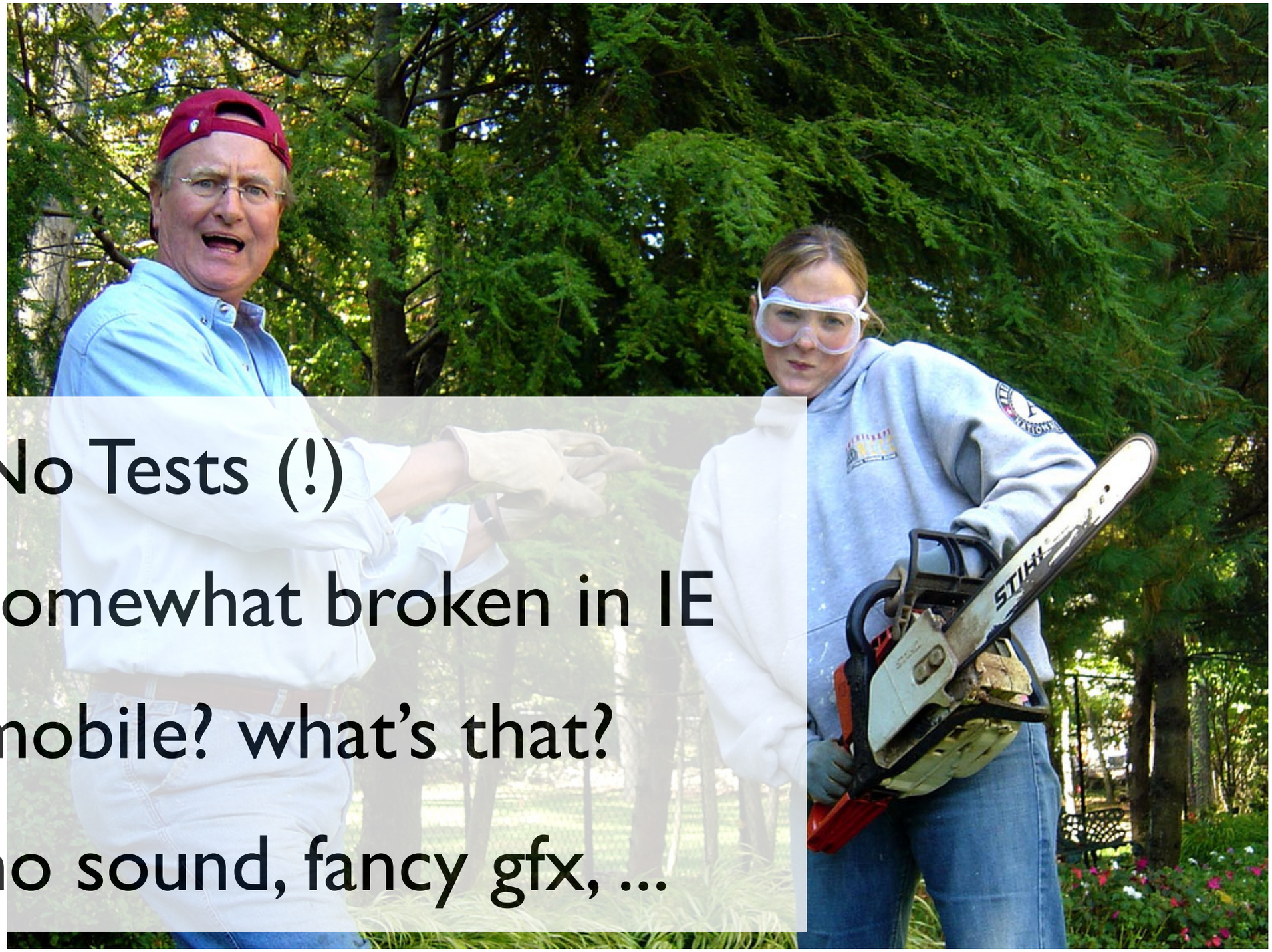


(but it's playable)

It's a Hack!

http://www.flickr.com/photos/cheesygarlicboy/269419718/

It's a Hack!



- No Tests (!)
- somewhat broken in IE
- mobile? what's that?
- no sound, fancy gfx, ...

http://www.flickr.com/photos/cheesygarlicboy/269419718/

If you see this...

you know what to expect.

What I'll cover...

- Basics of canvas 2D (as I go)
- Overview of how it's put together
- Some game mechanics
- Performance

What's Canvas?

- "New" to HTML5!
- lets you draw 2D graphics:
  - images, text
  - vector graphics (lines, curves, etc)
  - trades performance & control for convenience
- ... and 3D graphics (WebGL)
  - still a draft standard

Drawing a Ship

- Get canvas element
- draw lines that make up the ship

```
<body onload="drawShip()">
    <h1>Canvas:</h1>
    <canvas id="demo" width="300" height="200" style="border: 1px solid black" />
</body>
```

```
function drawShip() {
  var canvas = document.getElementById("demo");
  var ctx = canvas.getContext('2d');

  var center = {x: canvas.width/2, y: canvas.height/2};
  ctx.translate( center.x, center.y );

  ctx.strokeStyle = 'black';
  ctx.beginPath();
  ctx.moveTo(0,0);
  ctx.lineTo(14,7);
  ctx.lineTo(0,14);
  ctx.quadraticCurveTo(7,7, 0,0);
  ctx.closePath();
  ctx.stroke();
}
```

Canvas is an element.
You use one of its 'context' objects to draw to it.

2D Context is pretty simple

Walk through ctx calls:

- translate: move "origin" to center of canvas
- moveTo: move without drawing
- lineTo: draw a line
- curve: draw a curve

demo v

ship.html

syntax highlighting: http://tohtml.com

Moving it around

```
var canvas, ctx, center, ship;

function drawShipLoop() {
  canvas = document.getElementById("demo");
  ctx = canvas.getContext('2d');
  center = {x: canvas.width/2, y: canvas.height/2};
  ship = {x: center.x, y: center.y, facing: 0};

  setTimeout( updateAndDrawShip, 20 );
}

function updateAndDrawShip() {
  // set a fixed velocity:
  ship.y += 1;
  ship.x += 1;
  ship.facing += Math.PI/360 * 5;

  drawShip();

  if (ship.y < canvas.height-10) {
    setTimeout( updateAndDrawShip, 20 );
  } else {
    drawGameOver();
  }
}
```

```
function drawShip() {
  ctx.save();
  ctx.clearRect( 0,0, canvas.width,canvas.height );
  ctx.translate( ship.x, ship.y );
  ctx.rotate( ship.facing );

  ctx.strokeStyle = 'black';
  ctx.beginPath();
  ctx.moveTo(0,0);
  ctx.lineTo(14,7);
  ctx.lineTo(0,14);
  ctx.quadraticCurveTo(7,7, 0,0);
  ctx.closePath();
  ctx.stroke();

  ctx.restore();
}
```

```
function drawGameOver() {
  ctx.save();
  ctx.globalComposition = "lighter";
  ctx.font = "20px Verdana";
  ctx.fillStyle = "rgba(50,50,50,0.9)";
  ctx.fillText("Game Over", this.canvas.width/2 - 50,
               this.canvas.height/2);
  ctx.restore();
}
```

Introducing:

- animation loop: updateAndDraw...
- keeping track of an object's co-ords
- velocity & rotation
- clearing the canvas

Don't setInterval - we'll get to that later.
globalComposition - drawing mode, lighter so we can see text in some scenarios.

demo -->

ship-move.html

Controls

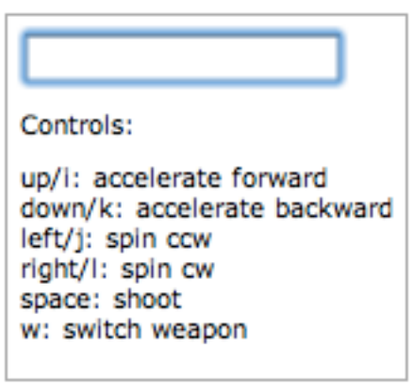
Wow!

```
<div id="controlBox">
    <input id="controls" type="text"
     placeholder="click to control" autofocus="autofocus"/>
    <p>Controls:
        <ul class="controlsInfo">
            <li>up/i: accelerate forward</li>
            <li>down/k: accelerate backward</li>
            <li>left/j: spin ccw</li>
            <li>right/l: spin cw</li>
            <li>space: shoot</li>
            <li>w: switch weapon</li>
        </ul>
    </p>
</div>
```

super-high-tech solution:

- use arrow keys to control your ship
- space to fire
- thinking of patenting it :)

```
$("#controls").keydown(function(event) {self.handleKeyEvent(event)});
$("#controls").keyup(function(event) {self.handleKeyEvent(event)});

AsteroidsGame.prototype.handleKeyEvent = function(event) {
    // TODO: send events, get rid of ifs.
    switch (event.which) {
      case 73: // i = up
      case 38: // up = accel
        if (event.type == 'keydown') {
            this.ship.startAccelerate();
        } else { // assume keyup
            this.ship.stopAccelerate();
        }
        event.preventDefault();
        break;
    ...
```



AsteroidsGame.js

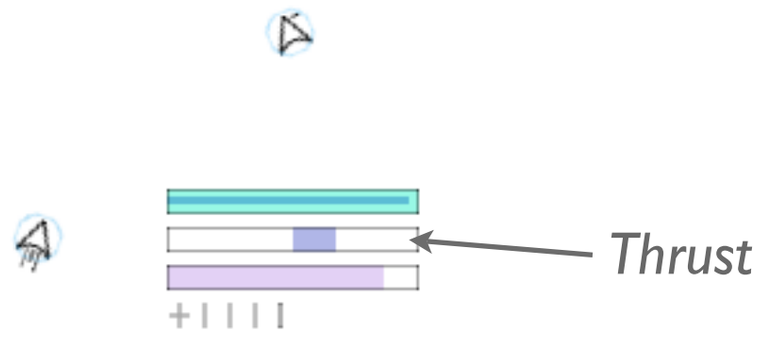
Controls: Feedback

- Lines: thrust forward, backward, or spin

  (think exhaust from a jet...)

- Thrust: 'force' in status bar.

```
// renderThrustForward
// offset from center of ship
// we translate here before drawing
render.x = -13;
render.y = -3;

ctx.strokeStyle = 'black';
ctx.beginPath();
ctx.moveTo(8,0);
ctx.lineTo(0,0);
ctx.moveTo(8,3);
ctx.lineTo(3,3);
ctx.moveTo(8,6);
ctx.lineTo(0,6);
ctx.closePath();
ctx.stroke();
```



Ships.js

Status bars...

... are tightly-coupled to Ships atm:

```
Ship.prototype.initialize = function(game, spatial) {
    // Status Bars
    // for displaying ship info: health, shield, thrust, ammo
    // TODO: move these into their own objects
    ...
    this.thrustWidth = 100;
    this.thrustHeight = 10;
    this.thrustX = this.healthX;
    this.thrustY = this.healthY + this.healthHeight + 5;
    this.thrustStartX = Math.floor( this.thrustWidth / 2 );
    ...

Ship.prototype.renderThrustBar = function() {
    var render = this.getClearThrustBarCanvas();
    var ctx = render.ctx;

    var thrustPercent = Math.floor(this.thrust/this.maxThrust * 100);
    var fillWidth = Math.floor(thrustPercent * this.thrustWidth / 100 / 2);
    var r = 100;
    var b = 200 + Math.floor(thrustPercent/2);
    var g = 100;
    var fillStyle = 'rgba('+ r +','+ g +','+ b +',0.5)';

    ctx.fillStyle = fillStyle;
    ctx.fillRect(this.thrustStartX, 0, fillWidth, this.thrustHeight);

    ctx.strokeStyle = 'rgba(5,5,5,0.75)';
    ctx.strokeRect(0, 0, this.thrustWidth, this.thrustHeight);

    this.render.thrustBar = render;
}
```

http://www.flickr.com/photos/imelda/497456854/

Drawing an Asteroid

(or planet)

```
ctx.beginPath();
ctx.arc(this.radius, this.radius, this.radius, 0, deg_to_rad[360], false);
ctx.closePath()
if (this.fillStyle) {
  ctx.fillStyle = this.fillStyle;
  ctx.fill();
} else {
  ctx.strokeStyle = this.strokeStyle;
  ctx.stroke();
}
```

Show: cookie cutter level

Planets.js

Compositing

Drawing an Asteroid

(or planet)

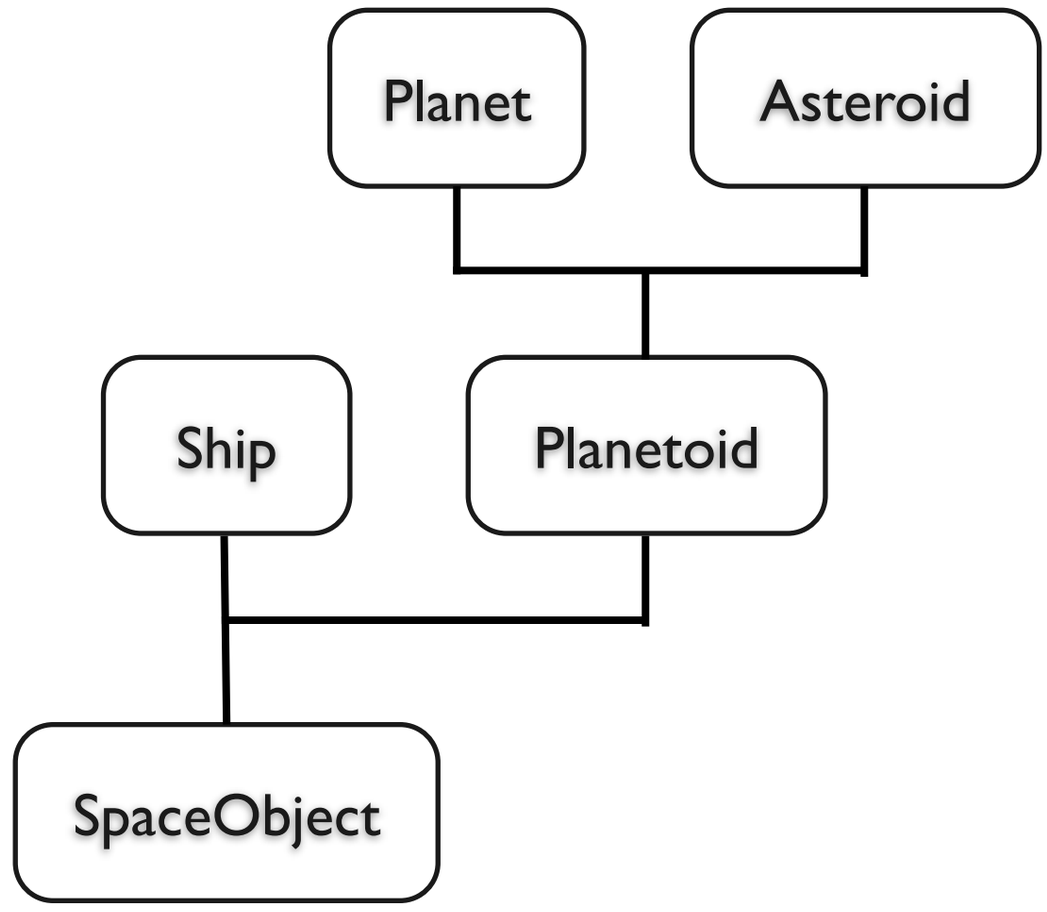
```
ctx.beginPath();
ctx.arc(this.radius, this.radius, this.radius, 0, deg_to_rad[360], false);
ctx.closePath()
if (this.fillStyle) {
  ctx.fillStyle = this.fillStyle;
  ctx.fill();
} else {
  ctx.strokeStyle = this.strokeStyle;
  ctx.stroke();
}
```

```
// Using composition as a cookie cutter:
if (this.image != null) {
  this.render = this.createPreRenderCanvas(this.radius*2, this.radius*2);
  var ctx = this.render.ctx;

  // Draw a circle to define what we want to keep:
  ctx.globalCompositeOperation = 'destination-over';
  ctx.beginPath();
  ctx.arc(this.radius, this.radius, this.radius, 0, deg_to_rad[360], false);
  ctx.closePath();
  ctx.fillStyle = 'white';
  ctx.fill();

  // Overlay the image:
  ctx.globalCompositeOperation = 'source-in';
  ctx.drawImage(this.image, 0, 0, this.radius*2, this.radius*2);
  return;
}
```

Show: cookie cutter level

Space Objects

- DRY
- Base class for all things spacey.
- Everything is a circle.



Err, why is everything a circle?

- An asteroid is pretty much a circle, right?
- And so are planets...
- And so is a Ship with a shield around it! ;-)

- ok, really:
  - Game physics can get complicated.
  - Keep it simple!

Space Objects

have...

- radius
- coords: x, y
- facing angle
- velocity, spin, thrust
- health, mass, damage
- and a whole lot more...

can...

- draw themselves
- update their positions
- accelerate, spin
- collide with other objects
- apply damage, die
- etc...

SpaceObject.js

Game Mechanics

what makes the game feel right.

Game Mechanics

what makes the game feel right.

This is where it gets hairy.

http://www.flickr.com/photos/maggz/2860543788/

The god class...

- AsteroidsGame does it all!
  - user controls, game loop, 95% of the game mechanics ...
  - Ok, so it’s not 5000 lines long (yet), but...
  - it should really be split up!

AsteroidsGame.js

Game Mechanics

- velocity & spin
- acceleration & drag
- gravity
- collision detection, impact, bounce
- health, damage, life & death
- object attachment & push
- out-of-bounds, viewports & scrolling

Velocity & Spin

```
SpaceObject.prototype.initialize = function(game,
spatial) {
   ...

    this.x = 0;      // starting position on x axis
    this.y = 0;      // starting position on y axis
    this.facing = 0; // currently facing angle (rad)

    this.stationary = false; // should move?

    this.vX = spatial.vX || 0; // speed along X axis
    this.vY = spatial.vY || 0; // speed along Y axis
    this.maxV = spatial.maxV || 2; // max velocity
    this.maxVSquared = this.maxV*this.maxV; // cache

    // thrust along facing
    this.thrust = spatial.initialThrust || 0;
    this.maxThrust = spatial.maxThrust || 0.5;
    this.thrustChanged = false;

    this.spin = spatial.spin || 0; // spin in Rad/sec
    this.maxSpin = deg_to_rad[10];
}

SpaceObject.prototype.updatePositions = function
(objects) {
    ...
    if (this.updateFacing(this.spin)) changed = true;
    if (this.updateX(this.vX)) changed = true;
    if (this.updateY(this.vY)) changed = true;
}
```

```
SpaceObject.prototype.updateX = function(dX) {
  if (this.stationary) return false;
  if (dX == 0) return false;
  this.x += dX;
  return true;
}

SpaceObject.prototype.updateY = function(dY) {
  if (this.stationary) return false;
  if (dY == 0) return false;
  this.y += dY;
  return true;
}

SpaceObject.prototype.updateFacing = function(delta)
{
  if (delta == 0) return false;
  this.facing += delta;

  // limit facing angle to 0 <= facing <= 360
  if (this.facing >= deg_to_rad[360] ||
      this.facing <= deg_to_rad[360]) {
    this.facing = this.facing % deg_to_rad[360];
  }

  if (this.facing < 0) {
    this.facing = deg_to_rad[360] + this.facing;
  }

  return true;
}
```

velocity = Δ distance / time

spin = angular velocity = Δ angle / time

Velocity & Spin

```
SpaceObject.prototype.initialize = function(game,
spatial) {
   ...

    this.x = 0;      // starting position on x axis
    this.y = 0;      // starting position on y axis
    this.facing = 0; // currently facing angle (rad)

    this.stationary = false; // should move?

    this.vX = spatial.vX || 0; // speed along X axis
    this.vY = spatial.vY || 0; // speed along Y axis
    this.maxV = spatial.maxV || 2; // max velocity
    this.maxVSquared = this.maxV*this.maxV; // cache

    // thrust along facing
    this.thrust = spatial.initialThrust || 0;
    this.maxThrust = spatial.maxThrust || 0.5;
    this.thrustChanged = false;

    this.spin = spatial.spin || 0; // spin in Rad/sec
    this.maxSpin = deg_to_rad[10];
}

SpaceObject.prototype.updatePositions = function
(objects) {
    ...
    if (this.updateFacing(this.spin)) changed = true;
    if (this.updateX(this.vX)) changed = true;
    if (this.updateY(this.vY)) changed = true;
}
```

```
SpaceObject.prototype.updateX = function(dX) {
  if (this.stationary) return false;
  if (dX == 0) return false;
  this.x += dX;
  return true;
}

SpaceObject.prototype.updateY = function(dY) {
  if (this.stationary) return false;
  if (dY == 0) return false;
  this.y += dY;
  return true;
}

SpaceObject.prototype.updateFacing = function(delta)
{
  if (delta == 0) return false;
  this.facing += delta;

  // limit facing angle to 0 <= facing <= 360
  if (this.facing >= deg_to_rad[360] ||
      this.facing <= deg_to_rad[360]) {
    this.facing = this.facing % deg_to_rad[360];
  }

  if (this.facing < 0) {
    this.facing = deg_to_rad[360] + this.facing;
  }

  return true;
}
```

velocity = Δ distance / time

spin = angular velocity = Δ angle / time

where: time = current frame rate

Acceleration

```
SpaceObject.prototype.initialize = function(game, spatial) {
   ...
    // thrust along facing
    this.thrust = spatial.initialThrust || 0;
    this.maxThrust = spatial.maxThrust || 0.5;
    this.thrustChanged = false;
}

SpaceObject.prototype.accelerateAlong = function(angle, thrust) {
    var accel = thrust/this.mass;
    var dX = Math.cos(angle) * accel;
    var dY = Math.sin(angle) * accel;
    this.updateVelocity(dX, dY);
}
```

acceleration = Δ velocity / time

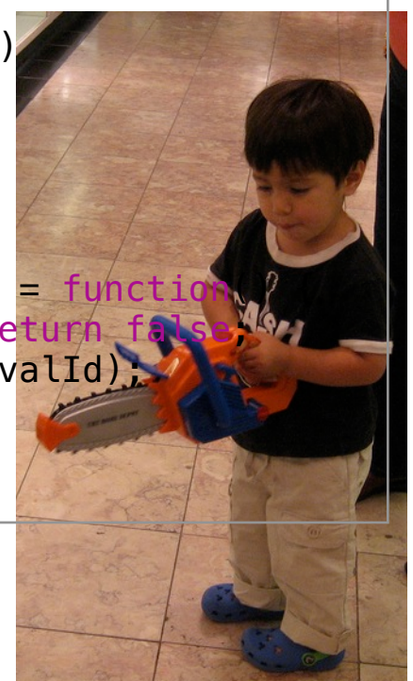
acceleration = mass / force

Acceleration

```
SpaceObject.prototype.initialize = function(game, spatial) {
   ...
    // thrust along facing
    this.thrust = spatial.initialThrust || 0;
    this.maxThrust = spatial.maxThrust || 0.5;
    this.thrustChanged = false;
}

SpaceObject.prototype.accelerateAlong = function(angle, thrust) {
    var accel = thrust/this.mass;
    var dX = Math.cos(angle) * accel;
    var dY = Math.sin(angle) * accel;
    this.updateVelocity(dX, dY);
}
```

```
Ship.prototype.startAccelerate = function() {
    if (this.accelerate) return;
    this.accelerate = true;
    //console.log("thrust++");

    this.clearSlowDownInterval();

    var self = this;
    this.incThrustIntervalId = setInterval(function(){
      self.increaseThrust();
    }, 20);  // real time
};

Ship.prototype.increaseThrust = function() {
    this.incThrust(this.thrustIncrement);
    this.accelerateAlong(this.facing, this.thrust);
}

Ship.prototype.stopAccelerate = function() {
    //console.log("stop thrust++");
    if (this.clearIncThrustInterval())
this.resetThrust();
    this.startSlowingDown();
    this.accelerate = false;
};

Ship.prototype.clearIncThrustInterval = function
    if (! this.incThrustIntervalId) return false;
    clearInterval(this.incThrustIntervalId);
    this.incThrustIntervalId = null;
    return true;
}
```

```
Ship.prototype.initialize = function(game, spatial) {
    ...

    spatial.mass = 10;

    // current state of user action:
    this.increaseSpin = false;
    this.decreaseSpin = false;
    this.accelerate = false;
    this.decelerate = false;
    this.firing = false;

    // for moving about:
    this.thrustIncrement = 0.01;
    this.spinIncrement = deg_to_rad[0.5];
    ...
}
```



acceleration = Δ velocity / time

acceleration = mass / force

Acceleration

```
SpaceObject.prototype.initialize = function(game, spatial) {
   ...
    // thrust along facing
    this.thrust = spatial.initialThrust || 0;
    this.maxThrust = spatial.maxThrust || 0.5;
    this.thrustChanged = false;
}

SpaceObject.prototype.accelerateAlong = function(angle, thrust) {
    var accel = thrust/this.mass;
    var dX = Math.cos(angle) * accel;
    var dY = Math.sin(angle) * accel;
    this.updateVelocity(dX, dY);
}
```

```
Ship.prototype.initialize = function(game, spatial) {
    ...

    spatial.mass = 10;

    // current state of user action:
    this.increaseSpin = false;
    this.decreaseSpin = false;
    this.accelerate = false;
    this.decelerate = false;
    this.firing = false;

    // for moving about:
    this.thrustIncrement = 0.01;
    this.spinIncrement = deg_to_rad[0.5];
    ...
}
```

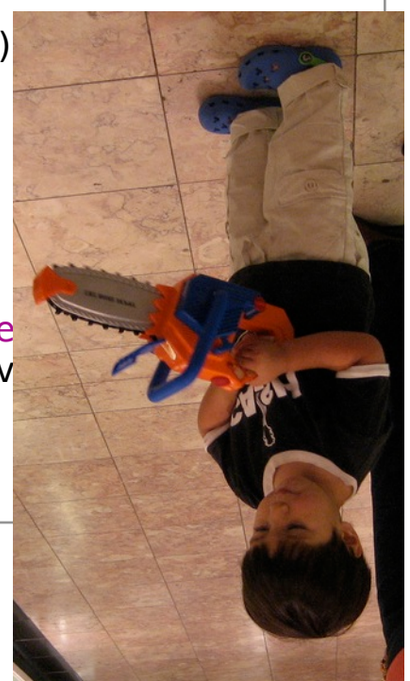
```
Ship.prototype.startAccelerate = function() {
    if (this.accelerate) return;
    this.accelerate = true;
    //console.log("thrust++");

    this.clearSlowDownInterval();

    var self = this;
    this.incThrustIntervalId = setInterval(function(){
      self.increaseThrust();
    }, 20);  // real time
};

Ship.prototype.increaseThrust = function() {
    this.incThrust(this.thrustIncrement);
    this.accelerateAlong(this.facing, this.thrust);
}


Ship.prototype.stopAccelerate = function() {
    //console.log("stop thrust++");
    if (this.clearIncThrustInterval())
this.resetThrust();
    this.startSlowingDown();
    this.accelerate = false;
};

Ship.prototype.clearIncThrustInterval
    if (! this.incThrustIntervalId) re
    clearInterval(this.incThrustInterv
    this.incThrustIntervalId = null;
    return true;
}
```

acceleration = Δ velocity / time

acceleration = mass / force

where: time = real time

(just to confuse things)

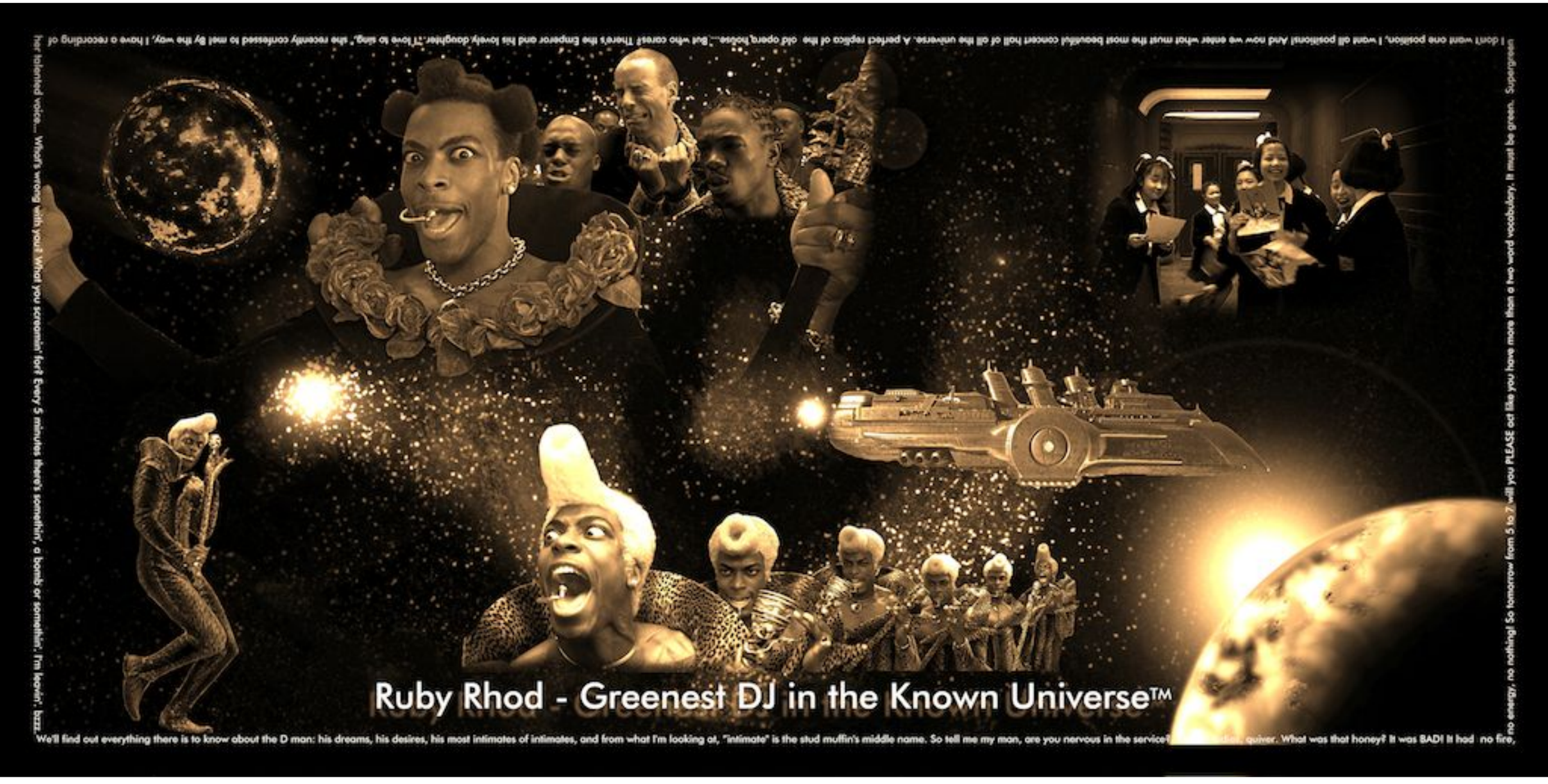


Drag

Yes, yes, there is no drag in outer space. Very clever.

I disagree.

Drag



http://nerdadjacent.deviantart.com/art/Ruby-Rhod-Supergreen-265156565

I disagree.

Drag

```
Ship.prototype.startSlowingDown = function() {
    // console.log("slowing down...");
    if (this.slowDownIntervalId) return;

    var self = this;
    this.slowDownIntervalId = setInterval(function(){
      self.slowDown()
    }, 100);  // eek! another hard-coded timeout!
}

Ship.prototype.clearSlowDownInterval = function() {
    if (! this.slowDownIntervalId) return false;
    clearInterval(this.slowDownIntervalId);
    this.slowDownIntervalId = null;
    return true;
}
```

```
Ship.prototype.slowDown = function() {
    var vDrag = 0.01;
    if (this.vX > 0) {
      this.vX -= vDrag;
    } else if (this.vX < 0) {
      this.vX += vDrag;
    }
    if (this.vY > 0) {
      this.vY -= vDrag;
    } else if (this.vY < 0) {
      this.vY += vDrag;
    }

    if (Math.abs(this.vX) <= vDrag) this.vX = 0;
    if (Math.abs(this.vY) <= vDrag) this.vY = 0;

    if (this.vX == 0 && this.vY == 0) {
      // console.log('done slowing down');
      this.clearSlowDownInterval();
    }
}
```

Demo: accel + drag in blank level

Gravity

```
var dvX_1 = 0, dvY_1 = 0;
if (! object1.stationary) {
  var accel_1 = object2.cache.G_x_mass / physics.dist_squared;
  if (accel_1 > 1e-5) { // skip if it's too small to notice
    if (accel_1 > this.maxAccel) accel_1 = this.maxAccel;
    var angle_1 = Math.atan2(physics.dX, physics.dY);
    dvX_1 = -Math.sin(angle_1) * accel_1;
    dvY_1 = -Math.cos(angle_1) * accel_1;
    object1.delayUpdateVelocity(dvX_1, dvY_1);
  }
}

var dvX_2 = 0, dvY_2 = 0;
if (! object2.stationary) {
  var accel_2 = object1.cache.G_x_mass / physics.dist_squared;
  if (accel_2 > 1e-5) { // skip if it's too small to notice
    if (accel_2 > this.maxAccel) accel_2 = this.maxAccel;
    // TODO: angle_2 = angle_1 - PI?
    var angle_2 = Math.atan2(-physics.dX, -physics.dY); // note the - signs
    dvX_2 = -Math.sin(angle_2) * accel_2;
    dvY_2 = -Math.cos(angle_2) * accel_2;
    object2.delayUpdateVelocity(dvX_2, dvY_2);
  }
}
```

force = G*mass1*mass2 / dist^2

acceleration1 = force / mass1

Collision Detection

```
AsteroidsGame.prototype.applyGamePhysicsTo = function(object1, object2) {
    ...
    var dX = object1.x - object2.x;
    var dY = object1.y - object2.y;

    // find dist between center of mass:
    // avoid sqrt, we don't need dist yet...
    var dist_squared = dX*dX + dY*dY;

    var total_radius = object1.radius + object2.radius;
    var total_radius_squared = Math.pow(total_radius, 2);

    // now check if they're touching:
    if (dist_squared > total_radius_squared) {
       // nope
    } else {
       // yep
       this.collision( object1, object2, physics );
    }

    ...
```

http://www.flickr.com/photos/wsmonty/4299389080/

Aren’t you glad we stuck with circles?

Bounce

Formula:

- *Don’t ask.*

bounce

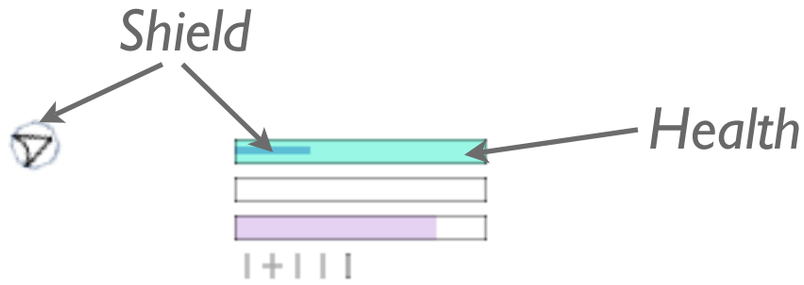
bounce

```
// Thanks Emanuelle! bounce algorithm adapted from:
// http://www.emanueleferonato.com/2007/08/19/managing-ball-vs-ball-collision-with-flash/
collision.angle = Math.atan2(collision.dY, collision.dX);
var magnitude_1 = Math.sqrt(object1.vX*object1.vX + object1.vY*object1.vY);
var magnitude_2 = Math.sqrt(object2.vX*object2.vX + object2.vY*object2.vY);

var direction_1 = Math.atan2(object1.vY, object1.vX);
var direction_2 = Math.atan2(object2.vY, object2.vX);

var new_vX_1 = magnitude_1*Math.cos(direction_1-collision.angle);
var new_vY_1 = magnitude_1*Math.sin(direction_1-collision.angle);
var new_vX_2 = magnitude_2*Math.cos(direction_2-collision.angle);
var new_vY_2 = magnitude_2*Math.sin(direction_2-collision.angle);

[snip]

// bounce the objects:
var final_vX_1 = ( (cache1.delta_mass * new_vX_1 + object2.cache.mass_x_2 * new_vX_2)
                 / cache1.total_mass * this.elasticity );
var final_vX_2 = ( (object1.cache.mass_x_2 * new_vX_1 + cache2.delta_mass * new_vX_2)
                 / cache2.total_mass * this.elasticity );
var final_vY_1 = new_vY_1 * this.elasticity;
var final_vY_2 = new_vY_2 * this.elasticity;


var cos_collision_angle = Math.cos(collision.angle);
var sin_collision_angle = Math.sin(collision.angle);
var cos_collision_angle_halfPI = Math.cos(collision.angle + halfPI);
var sin_collision_angle_halfPI = Math.sin(collision.angle + halfPI);

var vX1 = cos_collision_angle*final_vX_1 + cos_collision_angle_halfPI*final_vY_1;
var vY1 = sin_collision_angle*final_vX_1 + sin_collision_angle_halfPI*final_vY_1;
object1.delaySetVelocity(vX1, vY1);

var vX2 = cos_collision_angle*final_vX_2 + cos_collision_angle_halfPI*final_vY_2;
var vY2 = sin_collision_angle*final_vX_2 + sin_collision_angle_halfPI*final_vY_2;
object2.delaySetVelocity(vX2, vY2);
```

Aren’t you *really* glad we stuck with circles?

Making it *hurt*

```
AsteroidsGame.prototype.collision = function(object1, object2, collision) {
  ...
  // “collision” already contains a bunch of calcs
  collision[object1.id] = {
    cplane: {vX: new_vX_1, vY: new_vY_1}, // relative to collision plane
    dX: collision.dX,
    dY: collision.dY,
    magnitude: magnitude_1
  }
  // do the same for object2

  // let the objects fight it out
  object1.collided(object2, collision);
  object2.collided(object1, collision);
}
```




```
SpaceObject.prototype.collided = function(object, collision) {
    this.colliding[object.id] = object;

    if (this.damage) {
      var damageDone = this.damage;
      if (collision.impactSpeed != null) {
          damageDone = Math.ceil(damageDone * collision.impactSpeed);
      }
      object.decHealth( damageDone );
    }
}

SpaceObject.prototype.decHealth = function(delta) {
    this.healthChanged = true;
    this.health -= delta;
    if (this.health <= 0) {
      this.health = -1;
      this.die();
    }
}
```

```
Ship.prototype.decHealth = function(delta) {
    if (this.shieldActive) {
      delta = this.decShield(delta);
    }
    if (delta) Ship.prototype.parent.decHealth.call(this, delta);
}
```

When a collision occurs the Game Engine fires off 2 events to the objects in question

- For damage, I opted for a property rather than using mass * impact speed in the general case.

Applying damage is fairly straightforward:

- Objects are responsible for damaging each other
- When damage is done dec Health (for a Ship, shield first)
- If health < 0, an object dies.

SpaceObject.js

Object Lifecycle

```
SpaceObject.prototype.die = function() {
    this.died = true;
    this.update = false;
    this.game.objectDied( this );
}
```

```
AsteroidsGame.prototype.objectDied = function(object) {
    // if (object.is_weapon) {
    //} else if (object.is_asteroid) {
    if (object.is_planet) {
      throw "planet died!?"; // not allowed
    } else if (object.is_ship) {
      // TODO: check how many lives they've got
      if (object == this.ship) {
          this.stopGame();
      }
    }

    this.removeObject(object);
}

AsteroidsGame.prototype.removeObject = function(object) {
    var objects = this.objects;
    var i = objects.indexOf(object);
    if (i >= 0) {
      objects.splice(i,1);
      this.objectUpdated( object );
    }

    // avoid memory bloat: remove references to this object
    // from other objects' caches:
    var oid = object.id;
    for (var i=0; i < objects.length; i++) {
      delete objects[i].cache[oid];
    }
}
```

```
Asteroid.prototype.die = function() {
  this.parent.die.call( this );
  if (this.spawn <= 0) return;
  for (var i=0; i < this.spawn; i++) {
    var mass = Math.floor(this.mass / this.spawn * 1000)/1000;
    var radius = getRandomInt(2, this.radius);
    var asteroid = new Asteroid(this.game, {
        mass: mass,
        x: this.x + i/10, // don't overlap
        y: this.y + i/10,
        vX: this.vX * Math.random(),
        vX: this.vY * Math.random(),
        radius:  radius,
        health: getRandomInt(0, this.maxSpawnHealth),
        spawn: getRandomInt(0, this.spawn-1),
        image: getRandomInt(0, 5) > 0 ? this.image : null,
        // let physics engine handle movement
    });
    this.game.addObject( asteroid );
  }
}
```

```
AsteroidsGame.prototype.addObject = function(object) {
    //console.log('adding ' + object);
    this.objects.push( object );
    this.objectUpdated( object );
    object.preRender();
    this.cachePhysicsFor(object);
}
```

Attachment

- Attach objects that are ‘gently’ touching
  - then apply special physics
- Why?

AsteroidsGame.js

Attachment

- Attach objects that are ‘gently’ touching
  - then apply special physics
- Why?

Prevent the same collision from recurring.

+

Allows ships to land.

+

Poor man’s Orbit.

AsteroidsGame.js

Push!

- When objects get too close
  - push them apart!
  - otherwise they overlap...

(and the game physics gets weird)

demo: what happens when you disable **applyPushAway()**

Out-of-bounds

When you have a map that is *not* wrapped...

Simple strategy:

- kill most objects that stray
- push back important things like ships

```
AsteroidsGame.prototype.applyOutOfBounds = function(object) {
    if (object.stationary) return;

    var level = this.level;
    var die_if_out_of_bounds =
        !(object.is_ship || object.is_planet);

  if (object.x < 0) {
    if (level.wrapX) {
        object.setX(level.maxX + object.x);
    } else {
        if (die_if_out_of_bounds && object.vX < 0) {
          return object.die();
        }
        object.updateVelocity(0.1, 0);
    }
  } else if (object.x > level.maxX) {
    if (level.wrapX) {
        object.setX(object.x - level.maxX);
    } else {
        if (die_if_out_of_bounds && object.vX > 0) {
          return object.die();
        }
        object.updateVelocity(-0.1, 0);
    }
  }
  ...
```

```
  ...

  if (object.y < 0) {
    if (level.wrapY) {
        object.setY(level.maxY + object.y);
    } else {
        if (die_if_out_of_bounds && object.vY < 0) {
          return object.die();
        }
        // push back into bounds
        object.updateVelocity(0, 0.1);
    }
  } else if (object.y > level.maxY) {
    if (level.wrapY) {
        object.setY(object.y - level.maxY);
    } else {
        if (die_if_out_of_bounds && object.vY > 0) {
          return object.die();
        }
        // push back into bounds
        object.updateVelocity(0, -0.1);
    }
  }
}
```

Viewport + Scrolling

When the dimensions of your map exceed those of your canvas...

```
AsteroidsGame.prototype.updateViewOffset = function() {
    var canvas = this.ctx.canvas;
    var offset = this.viewOffset;
    var dX = Math.round(this.ship.x - offset.x - canvas.width/2);
    var dY = Math.round(this.ship.y - offset.y - canvas.height/2);

    // keep the ship centered in the current view, but don't let the view
    // go out of bounds
    offset.x += dX;
    if (offset.x < 0) offset.x = 0;
    if (offset.x > this.level.maxX-canvas.width) offset.x = this.level.maxX-canvas.width;

    offset.y += dY;
    if (offset.y < 0) offset.y = 0;
    if (offset.y > this.level.maxY-canvas.height) offset.y = this.level.maxY-canvas.height;
}
```

Let browser manage complexity: if you draw to canvas outside of current width/height, browser doesn't draw it.

```
AsteroidsGame.prototype.redrawCanvas = function() {
...
    // shift view to compensate for current offset
    var offset = this.viewOffset;
    ctx.save();
    ctx.translate(-offset.x, -offset.y);
```

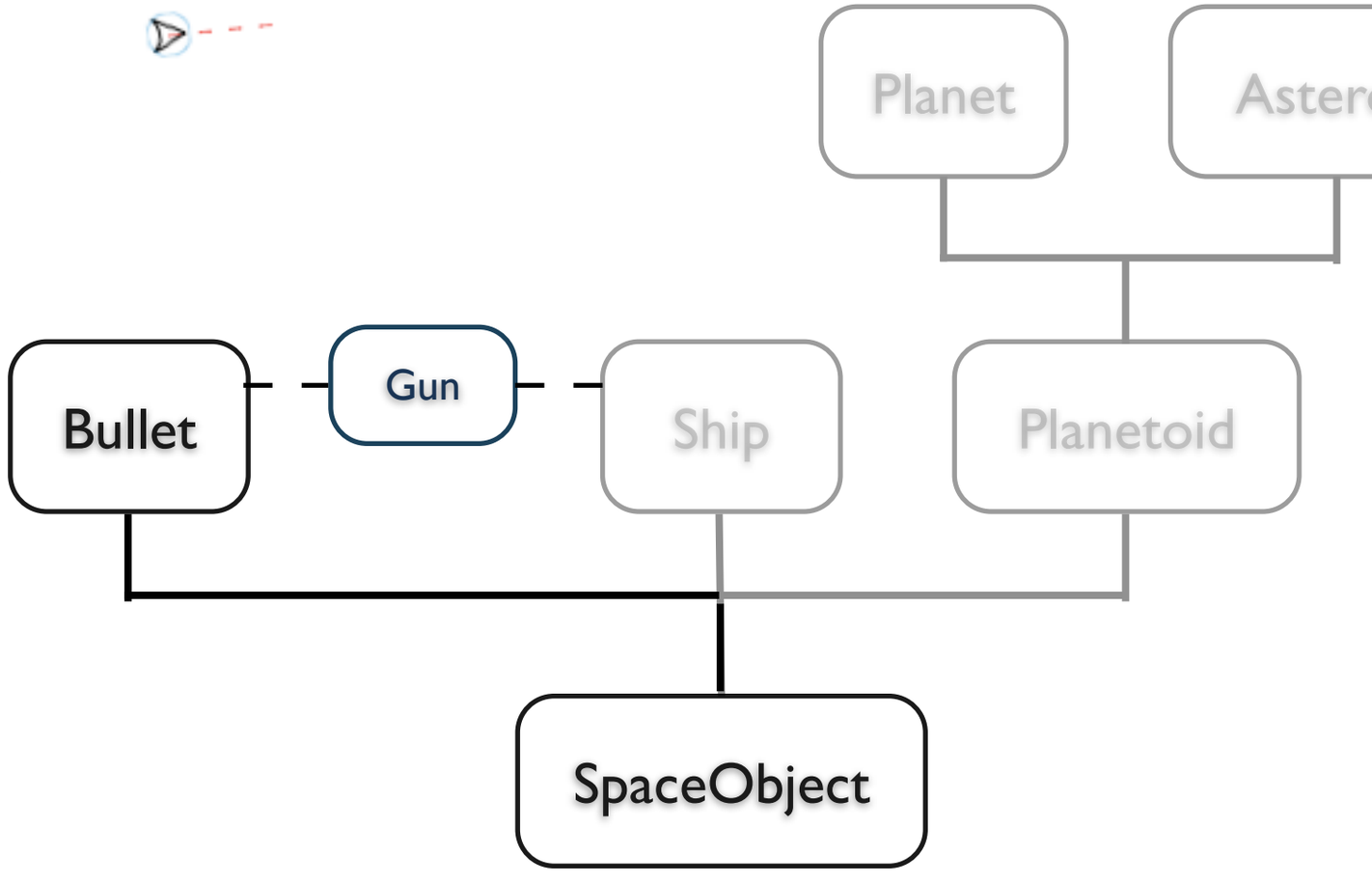
Putting it all together

Demo: hairballs & chainsaws level.

Weapons

Gun + Bullet

- Gun
  - Fires Bullets
  - Has ammo
  - Belongs to a Ship

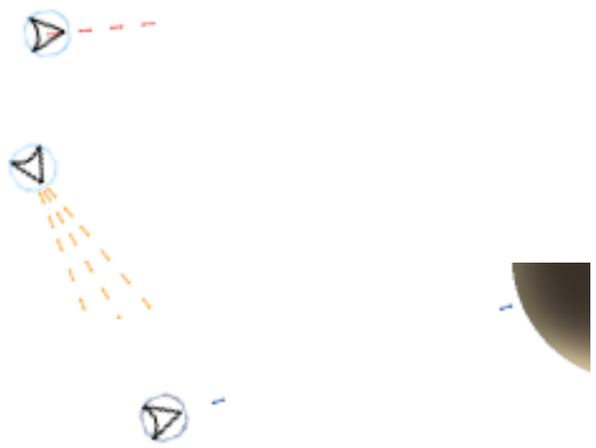


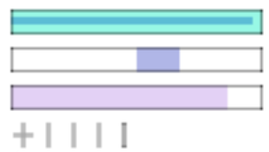
When you shoot, bullets inherit the Ship's velocity.
Each weapon has a different recharge rate (measured in real time).

Weapons.js

Other Weapons

- Gun
- SprayGun
- Cannon
- GrenadeCannon
- GravBenda™





Current weapon

Enemies.

(because shooting circles gets boring)

A basic enemy...

```
ComputerShip.prototype.findAndDestroyClosestEnemy = function() {
    var enemy = this.findClosestEnemy();
    if (enemy == null) return;

    // Note: this is a basic algorith, it doesn't take a lot of things
    // into account (enemy trajectory & facing, other objects, etc)

    // navigate towards enemy
    // shoot at the enemy
}
```

Demo: Level Lone enemy.
Show: ComputerShip class...

Ships.js

A basic enemy...

```
ComputerShip.prototype.findAndDestroyClosestEnemy = function() {
    var enemy = this.findClosestEnemy();
    if (enemy == null) return;

    // Note: this is a basic algorith, it doesn't take a lot of things
    // into account (enemy trajectory & facing, other objects, etc)

    // navigate towards enemy
    // shoot at the enemy
}
```

of course, it's a bit more involved...

Demo: Level Lone enemy.
Show: ComputerShip class...

Ships.js

Levels

- Define:
  - map dimensions
  - space objects
  - spawning
  - general properties of the canvas - color, etc

Levels.js

```
/*****************************************************************************
 * TrainingLevel: big planet out of field of view with falling asteroids.
 */

function TrainingLevel(game) {
    if (game) return this.initialize(game);
    return this;
}

TrainingLevel.inheritsFrom( Level );
TrainingLevel.description = "Training Level - learn how to fly!";
TrainingLevel.images = [ "planet.png", "planet-80px-green.png" ];

gameLevels.push(TrainingLevel);

TrainingLevel.prototype.initialize = function(game) {
    TrainingLevel.prototype.parent.initialize.call(this, game);
    this.wrapX = false;
    this.wrapY = false;

    var maxX = this.maxX;
    var maxY = this.maxY;

    var canvas = this.game.ctx.canvas;
    this.planets.push(
    {x: 1/2*maxX, y: 1/2*maxY, mass: 100, radius: 50, damage: 5, stationary: true, image_src: "planet.png" }
    , {x: 40, y: 40, mass: 5, radius: 20, vX: 2, vY: 0, image_src:"planet-80px-green.png"}
    , {x: maxX-40, y: maxY-40, mass: 5, radius: 20, vX: -2, vY: 0, image_src:"planet-80px-green.png"}
    );

    this.ships.push(
    {x: 4/5*canvas.width, y: 1/3*canvas.height}
    );

    this.asteroids.push(
    {x: 1/10*maxX, y: 6/10*maxY, mass: 0.5, radius: 14, vX: 0, vY: 0, spawn: 1, health: 1},
        {x: 1/10*maxX, y: 2/10*maxY, mass: 1, radius: 5, vX: 0, vY: -0.1, spawn: 3 },
        {x: 5/10*maxX, y: 1/10*maxY, mass: 2, radius: 6, vX: -0.2, vY: 0.25, spawn: 4 },
        {x: 5/10*maxX, y: 2/10*maxY, mass: 3, radius: 8, vX: -0.22, vY: 0.2, spawn: 7 }
    );
}
```

As usual, I had grandiose plans of an interactive level editor... This was all I had time for.

Levels.js

Performance

"Premature optimisation is the root of all evil."

http://c2.com/cgi/wiki?PrematureOptimization

Use requestAnimationFrame

Paul Irish knows why:

- don’t animate if your canvas is not visible
- adjust your frame rate based on actual performance
- lets the browser manage your app better

Profile your code

- profile in different browsers
- identify the slow stuff
- ask yourself: *"do we really need to do this?"*
- optimise it?
    - cache slow operations
    - change algorithm?
    - simplify?

Examples...

```
// see if we can use cached values first:
var g_cache1 = physics.cache1.last_G;
var g_cache2 = physics.cache2.last_G;

if (g_cache1) {
  var delta_dist_sq = Math.abs( physics.dist_squared - g_cache1.last_dist_squared);
  var percent_diff = delta_dist_sq / physics.dist_squared;
  // set threshold @ 5%
  if (percent_diff < 0.05) {
    // we haven't moved much, use last G values
    //console.log("using G cache");
    object1.delayUpdateVelocity(g_cache1.dvX, g_cache1.dvY);
    object2.delayUpdateVelocity(g_cache2.dvX, g_cache2.dvY);
    return;
  }
}
```

```
// avoid overhead of update calculations & associated checks: batch together
SpaceObject.prototype.delayUpdateVelocity = function(dvX, dvY) {
    if (this._updates == null) this.init_updates();
    this._updates.dvX += dvX;
    this._updates.dvY += dvY;
}
```

```
var dist_squared = dX*dX + dY*dY; // avoid sqrt, we don't need dist yet
```

```
// put any calculations we can avoid repeating here
AsteroidsGame.prototype.cachePhysicsFor = function(object1) {
    for (var i=0; i < this.objects.length; i++) {
       var object2 = this.objects[i];
       if (object1 == object2) continue;

       // shared calcs
       var total_radius = object1.radius + object2.radius;
       var total_radius_squared = Math.pow(total_radius, 2);
       var total_mass = object1.mass + object2.mass;

       // create separate caches from perspective of objects:
       object1.cache[object2.id] = {
           total_radius: total_radius,
           total_radius_squared: total_radius_squared,
           total_mass: total_mass,
           delta_mass: object1.mass - object2.mass
       }

       object2.cache[object1.id] = {
           total_radius: total_radius,
           total_radius_squared: total_radius_squared,
           total_mass: total_mass,
           delta_mass: object2.mass - object1.mass
       }
    }
}
```

```
this.maxVSquared = this.maxV*this.maxV;   // cache for speed
```

```
if (accel_1 > 1e-5) { // skip if it's too small to notice
```

```
...
```

Performance

Great Ideas from Boris Smus:

- Only redraw changes
- Pre-render to another canvas
- Draw background in another canvas / element
- Don't use floating point co-ords

... and more ...

Can I Play?

http://www.spurkis.org/asteroids/

See Also...

Learning / examples:

- https://developer.mozilla.org/en-US/docs/Canvas_tutorial
- http://en.wikipedia.org/wiki/Canvas_element
- http://www.html5rocks.com/en/tutorials/canvas/performance/
- http://www.canvasdemos.com
- http://billmill.org/static/canvastutorial/
- http://paulirish.com/2011/requestanimationframe-for-smart-animating/

Specs:

- http://dev.w3.org/html5/spec/
- http://www.khronos.org/registry/webgl/specs/latest/

Questions?