Developing Games Using Data not Trees

Drew Petersen
@KirbySaysHi

ONE MAN

VS HIS OWN MIND

24 HOURS

THIS

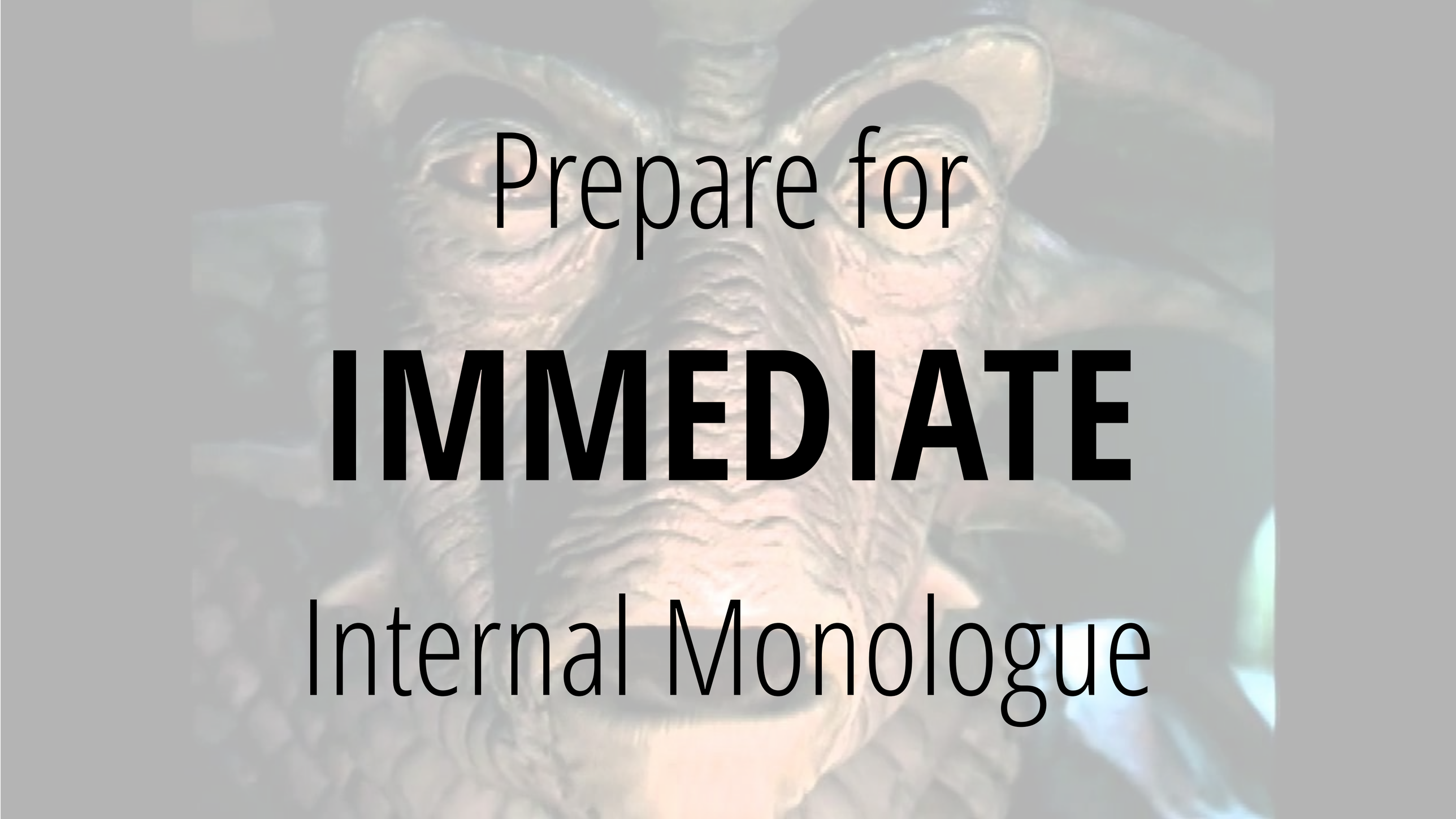
IS

A GAME JAM

There is just one problem…

ASTEROIDS: TOTALLY DIFFERENT THIS TIME

is not actually different.



Prepare for

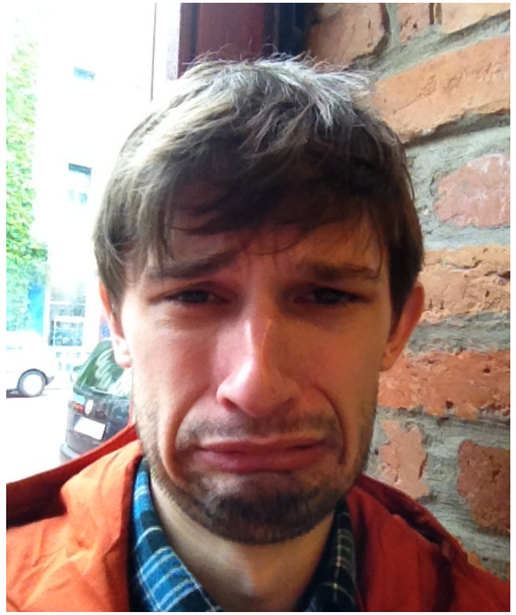
IMMEDIATE

Internal Monologue

Oh no! It’s not different at all!

But it uses typed arrays and verlet physics and all this stuff that I had a lot of fun making!

But the players don’t actually care about that…



What am I going to do? There are only a few hours left before the deadline!

“Maybe I can quickly change it.”

–Every programmer under a deadline, ever.

Maybe if the ship is huge?

Maybe if the asteroids are tiny?

“Boring.”

–Me, to myself

What if…

you...

…control the asteroids?

How will I have time?

Oh right.

I modeled everything as data.

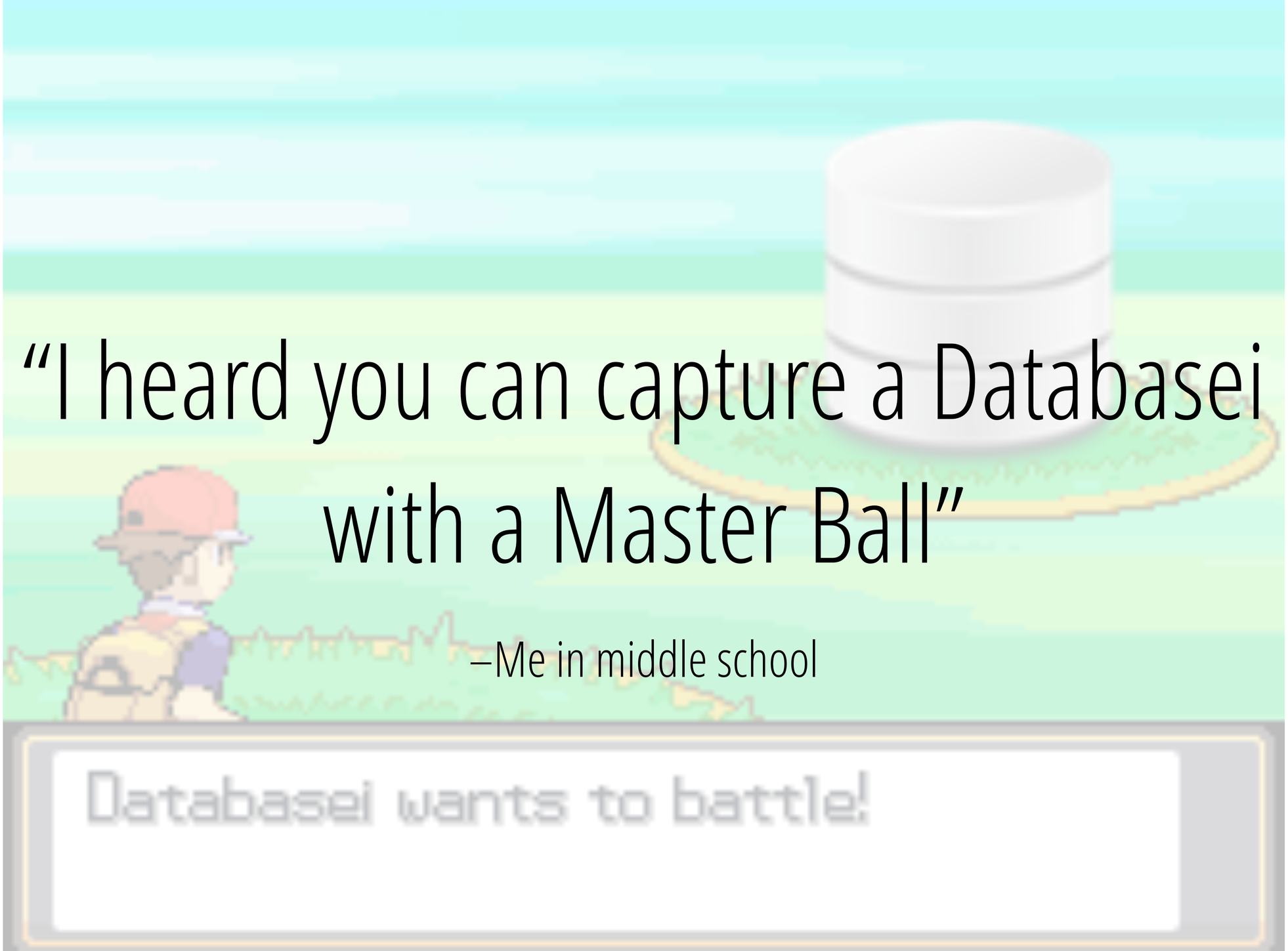
Wait, what does that mean?

Is that like a database?

Databases are fun!

```
SELECT *
FROM boring
LIMIT fun
```

Databases are not a game.



“I heard you can capture a Databasei with a Master Ball”

–Me in middle school

How does one data?

Does data you?

Data Matthews Band really replicates.

Base into data without remorse

OH GAWD I’VE DEVOLVED TO MEME

But seriously, how?

Approach: Object Oriented

Objects

- Ship
- Asteroid
- Bullet

```
├ WorldEntity
 ├ PhysicsEntity
  ├ CollidableEntity
   ├ RenderableEntity
    ├ AsteroidEntity
```

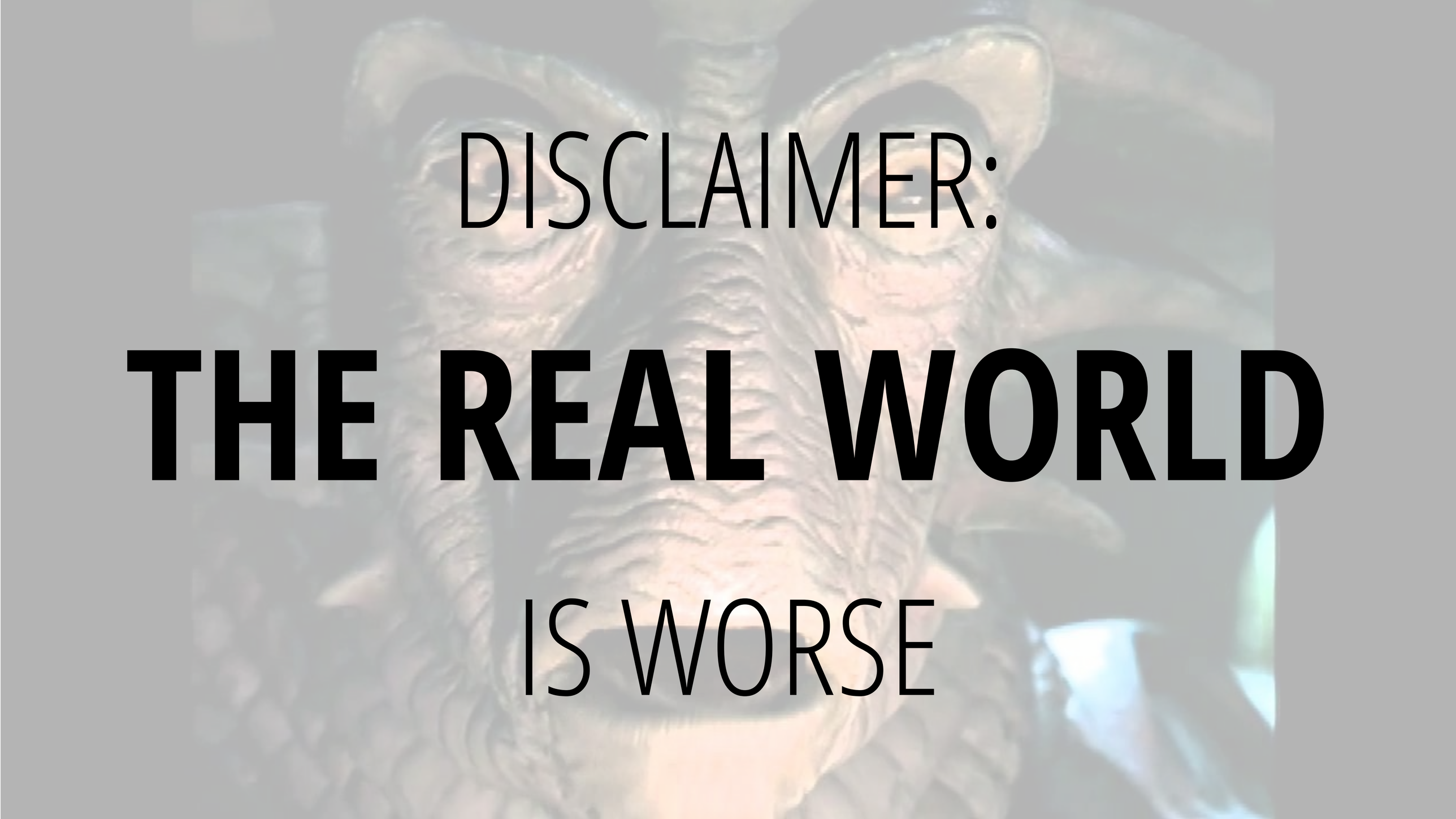
```
├ WorldEntity
 ├ PhysicsEntity
  ├ CollidableEntity
   ├ RenderableEntity
    ├ BulletEntity
```

```
├ WorldEntity
 ├ PhysicsEntity              update() (WorldE)
  ├ CollidableEntity          handleCollisions() (CE)
   ├ RenderableEntity         draw() (RenderableE)
    ├ KeyboardCtrlEntity      handleInput() (KBCE)
     ├ ShipEntity             shoot()
```

What about controllable asteroids?

```
├ WorldEntity
 ├ PhysicsEntity                update() (WorldE)
  ├ CollidableEntity            handleCollisions() (CE)
   ├ RenderableEntity           draw() (RenderableE)
    ├ KeyboardCtrlEntity        handleInput() (KBCE)
     ├ ShipEntity               shoot()
```

AsteroidEntity



DISCLAIMER:

THE REAL WORLD

IS WORSE

ProCon: Professional Conventions for Professionals

- Logic appears to be local
- State is spread throughout the hierarchy
- Hard to optimize (batch all physics, batch draw, etc)

Approach: Data Oriented

- Functions operate on data, not Objects.
- There will always be more than one.
- Don't inhibit optimization (batching).

DISCLAIMER:

IT'S NOT PERFECT

But Might Be Better.

Starts with a Pocket

```
var pkt = new Pocket()
pkt.tick(16); // ms
```

We add Component Types to the Pocket

“Components like in ReactJS?”

–Ben Newman, probably

NOPE

```
pkt.componentType('rotation', function(cmp, opts) {
  cmp.angle = opts.angle || 0;
  cmp.rate = opts.rate || 0;
})

// shortcut / alias
pkt.cmpType('rotation', function(cmp, opts) {
  cmp.angle = opts.angle || 0;
  cmp.rate = opts.rate || 0;
})
```

```
pkt.cmpType('drag', function(cmp, opts) {
  cmp.percentage = opts.percentage || 0.99;
})
```

We request a
Key
from the Pocket

```
var key = pkt.key({
  'rotation': { rate: 0.9 },
  'drag': null
})

console.log(key); // 1
console.log(typeof key); // number
```

```
pkt.key({
  'ship': null,
  'human-controlled-01': null,
  'verlet-position': {
    x: pkt.firstData('ctx-2d').center.x,
    y: pkt.firstData('ctx-2d').center.y
  },
  'rotation': { rate: 0.1 },
  'thrust': null,
  'drag': null,
  'projectile-launcher': { launchForce: 10 },
  'point-shape': { points: [
    { x: size, y: 0 },
    { x: -size, y: -size / 2 },
    { x: -size, y:  size / 2 }
  ]},
  'bbox': null
})
```

We write
Systems
to modify
Component Data

```
pkt.system('input-thrust',
  ['verlet-position', 'rotation',
   'thrust', 'human-controlled-01'],
  function(pkt, keys, positions, rotations, thrusts) {
    for (var i = 0; i < keys.length; i++) {
      // Do something
    }
  })
```

Name

```
pkt.system('input-thrust',
  ['verlet-position', 'rotation',
   'thrust', 'human-controlled-01'],
  function(pkt, keys, positions, rotations, thrusts) {
    for (var i = 0; i < keys.length; i++) {
      // Do something
    }
  })
```

Name

Component Type Requirements

```
pkt.system('input-thrust',
  ['verlet-position', 'rotation',
   'thrust', 'human-controlled-01'],
  function(pkt, keys, positions, rotations, thrusts) {
    for (var i = 0; i < keys.length; i++) {
      // Do something
    }
  })
```

Name

Component Type

Requirements

```
pkt.system('input-thrust',
  ['verlet-position', 'rotation',
   'thrust', 'human-controlled-01'],
  function(pkt, keys, positions, rotations, thrusts) {
    for (var i = 0; i < keys.length; i++) {
      // Do something
    }
  })
```

Action

Name

Component Type Requirements

```
pkt.system('input-thrust',
  ['verlet-position', 'rotation',
   'thrust', 'human-controlled-01'],
  function(pkt, keys, positions, rotations, thrusts) {
    for (var i = 0; i < keys.length; i++) {
      // Do something
    }
  })
```

Action

Array of Keys that each have the required Component Datas

Name

Component Type Requirements

```
pkt.system('input-thrust',
  ['verlet-position', 'rotation',
   'thrust', 'human-controlled-01'],
  function(pkt, keys, positions, rotations, thrusts) {
    for (var i = 0; i < keys.length; i++) {
      // Do something
    }
  })
```

Action

Array of Keys that each have the required Component Datas

Global Collection of Component Data by Key

```
pkt.system(
  'render-point-shape',
  ['verlet-position', 'point-shape', 'rotation'],
  function(pkt, keys, positions, shapes, rotations) {
    var ctx2d = pkt.firstData('ctx-2d')

    for (var i = 0; i < keys.length; i++) {
      var k = keys[i];
      var position = positions[k];
      var shape = shapes[k];
      var rotation = rotations[k];
   }
  }
)
```

```
pkt.system(
  'render-point-shape',
  ['verlet-position', 'point-shape', 'rotation'],
  function(pkt, keys, positions, shapes, rotations) {
    var ctx2d = pkt.firstData('ctx-2d')

    for (var i = 0; i < keys.length; i++) {
      var k = keys[i];
      var position = positions[k];
      var shape = shapes[k];
      var rotation = rotations[k];
   }
  }
)
```

Manually grab each component data from the global collections.

```
pkt.systemForEach('input-thrust',
  ['verlet-position', 'rotation',
   'thrust', 'human-controlled-01'],
  function(pkt, key, position, rotation, thrust) {
    // Do Something
  })
```

```
pkt.systemForEach('input-thrust',
  ['verlet-position', 'rotation',
   'thrust', 'human-controlled-01'],
  function(pkt, key, position, rotation, thrust) {
    // Do Something
  })
```

Individual Key

```
pkt.systemForEach('input-thrust',
  ['verlet-position', 'rotation',
   'thrust', 'human-controlled-01'],
  function(pkt, key, position, rotation, thrust) {
    // Do Something
  })
```

Individual Key

Component Data
Matching the Key

“But what about player controlled asteroids?”

–Maybe one person in the audience?

Oh right.

```
pkt.systemForEach(
  'input-rotation',
  ['rotation', 'human-controlled-01'],
  function(pkt, key, rotation) {
    var input = pkt.firstData('keyboard-state');

    if (input.down.RIGHT) {
      rotation.angle += rotation.rate;
    } else if (input.down.LEFT) {
      rotation.angle -= rotation.rate;
    }
  }
)
```

```
pkt.systemForEach(
  'input-rotation',
  ['rotation', 'human-controlled-01'],
  function(pkt, key, rotation) {
    var input = pkt.firstData('keyboard-state');

    if (input.down.RIGHT) {
      rotation.angle += rotation.rate;
    } else if (input.down.LEFT) {
      rotation.angle -= rotation.rate;
    }
  }
)
```

```
pkt.systemForEach(
  'input-rotation',
  ['rotation', 'human-controlled-01'],
  function(pkt, key, rotation) {
    var input = pkt.firstData('keyboard-state');

    if (input.down.RIGHT) {
      rotation.angle += rotation.rate;
    } else if (input.down.LEFT) {
      rotation.angle -= rotation.rate;
    }
  }
)
```

?

```
pkt.systemForEach(
  'input-rotation',
  ['rotation', 'human-controlled-01'],
  function(pkt, key, rotation) {
    var input = pkt.firstData('keyboard-state');

    if (input.down.RIGHT) {
      rotation.angle += rotation.rate;
    } else if (input.down.LEFT) {
      rotation.angle -= rotation.rate;
    }
  }
)
```

?

Why no human-controlled-01 component data?

THERE IS NO

human-controlled-01

COMPONENT

We use
Labels
as anonymous components

```
pkt.key({
  'asteroid': null,
  'verlet-position': {
    x: x,
    y: y,
    acel: v2(acelX, acelY) },
  'point-shape': { points: points },
  'bbox': null,
  'rotation': null
})
```

There is no 'asteroid' component

```
pkt.system(
  'asteroid-ship-collider',
  ['point-shape', 'verlet-position',
   'rotation', 'asteroid'],
  function(pkt, keys, shapes, positions, rotations) {
  })
```

```
pkt.system(
  'asteroid-ship-collider',
  ['point-shape', 'verlet-position',
   'rotation', 'asteroid'],
  function(pkt, keys, shapes, positions, rotations) {
  })
```

We only want asteroids in here.

Ship also has point-shape, verlet-position, and rotation

Inspector | Console | Debugger | Style Editor | Canvas | Performance | Network

Net | CSS | JS | Security | Logging | Clear | Filter output

```
"Found no component initializer for "input", assuming it is a label."        bundle.js:833
"Found no component initializer for "ship", assuming it is a label."         bundle.js:833
"Found no component initializer for "human-controlled-01", assuming it is a label."   bundle.js:833
"Found no component initializer for "asteroid", assuming it is a label."     bundle.js:833
```

```
pkt.systemForEach(
  'input-rotation',
  ['rotation', 'human-controlled-01'],
  function(pkt, key, rotation) {
    var input = pkt.firstData('keyboard-state');

    if (input.down.RIGHT) {
      rotation.angle += rotation.rate;
    } else if (input.down.LEFT) {
      rotation.angle -= rotation.rate;
    }
  }
)
```

```
pkt.systemForEach(
  'input-rotation',
  ['rotation', 'human-controlled-01'],
  function(pkt, key, rotation) {
    var input = pkt.firstData('keyboard-state');

    if (input.down.RIGHT) {
      rotation.angle += rotation.rate;
    } else if (input.down.LEFT) {
      rotation.angle -= rotation.rate;
    }
  }
)
```

```
pkt.systemForEach(
  'input-rotation',
  ['rotation', 'human-controlled-01'],
  function(pkt, key, rotation) {
    var input = pkt.firstData('keyboard-state');

    if (input.down.RIGHT) {
      rotation.angle += rotation.rate;
    } else if (input.down.LEFT) {
      rotation.angle -= rotation.rate;
    }
  }
)
```

?

What happens if we add that label to the asteroids?

```
pkt.key({
  'asteroid': null,
  'human-controlled-01': null,
  'verlet-position': {
    x: x,
    y: y,
    acel: v2(acelX, acelY) },
  'point-shape': { points: points },
  'bbox': null,
  'rotation': null
})
```

Nothing?

```
pkt.systemForEach(
  'input-rotation',
  ['rotation', 'human-controlled-01'],
  function(pkt, key, rotation) {
    var input = pkt.firstData('keyboard-state');

    if (input.down.RIGHT) {
      rotation.angle += rotation.rate;
    } else if (input.down.LEFT) {
      rotation.angle -= rotation.rate;
    }
  }
)
```

```
pkt.systemForEach(
  'input-rotation',
  ['rotation', 'human-controlled-01'],
  function(pkt, key, rotation) {
    var input = pkt.firstData('keyboard-state');

    if (input.down.RIGHT) {
      rotation.angle += rotation.rate;
    } else if (input.down.LEFT) {
      rotation.angle -= rotation.rate;
    }
  }
)
```

```
pkt.systemForEach(
  'input-rotation',
  ['rotation', 'human-controlled-01'],
  function(pkt, key, rotation) {
    var input = pkt.firstData('keyboard-state');

    if (input.down.RIGHT) {
      rotation.angle += rotation.rate;
    } else if (input.down.LEFT) {
      rotation.angle -= rotation.rate;
    }
  }
)
```

```
pkt.cmpType('rotation', function(cmp, opts) {
  cmp.angle = opts.angle || 0;
  cmp.rate = opts.rate || 0;
})
```

```
pkt.cmpType('rotation', function(cmp, opts) {
  cmp.angle = opts.angle || 0;
  cmp.rate = opts.rate || 0;
})
```

```
pkt.key({
  'asteroid': null,
  'human-controlled-01': null,
  'verlet-position': {
    x: x,
    y: y,
    acel: v2(acelX, acelY) },
  'point-shape': { points: points },
  'bbox': null,
  'rotation': null
})
```

```
pkt.key({
  'asteroid': null,
  'human-controlled-01': null,
  'verlet-position': {
    x: x,
    y: y,
    acel: v2(acelX, acelY) },
  'point-shape': { points: points },
  'bbox': null,
  'rotation': { rate: 0.1 }
})
```

They rotate!

Let's make them thrust.

```
pkt.systemForEach(
  'input-thrust',
  ['verlet-position', 'rotation', 'thrust', 'human-controlled-01'],
  function(pkt, key, position, rotation, thrust) {
    var input = pkt.firstData('keyboard-state');

    if (input.down.UP) {
      var x = Math.cos(rotation.angle) * thrust.force;
      var y = Math.sin(rotation.angle) * thrust.force;
      position.acel.x += x;
      position.acel.y += y;
    }
  }
)
```

```
pkt.systemForEach(
  'input-thrust',
  ['verlet-position', 'rotation', 'thrust', 'human-controlled-01'],
  function(pkt, key, position, rotation, thrust) {
    var input = pkt.firstData('keyboard-state');

    if (input.down.UP) {
      var x = Math.cos(rotation.angle) * thrust.force;
      var y = Math.sin(rotation.angle) * thrust.force;
      position.acel.x += x;
      position.acel.y += y;
    }
  }
)
```

```
'human-controlled-01': null,
'verlet-position': {
  x: pkt.firstData('ctx-2d').center.x,
  y: pkt.firstData('ctx-2d').center.y
},
'rotation': { rate: 0.1 },
'thrust': null,
'drag': null,
'projectile-launcher': { launchForce: 10 },
'point-shape': { points: [
  { x: size, y: 0 },
  { x: -size, y: -size / 2 },
  { x: -size, y:  size / 2 }
]},
```

```
pkt.key({
  'asteroid': null,
  'human-controlled-01': null,
  'thrust': null,
  'verlet-position': {
    x: x,
    y: y,
    acel: v2(acelX, acelY) },
  'point-shape': { points: points },
  'bbox': null,
  'rotation': { rate: 0.1 },
})
```

Let's make them shoot!

```
pkt.key({
  'asteroid': null,
  'human-controlled-01': null,
  'thrust': null,
  'projectile-launcher': { launchForce: 10 },
  'verlet-position': {
    x: x,
    y: y,
    acel: v2(acelX, acelY) },
  'point-shape': { points: points },
  'bbox': null,
  'rotation': { rate: 0.1 },
})
```

Let's make the ship get hit!

```
pkt.system(
  'asteroid-projectile-collider',
  ['point-shape', 'verlet-position',
   'rotation', 'asteroid'],
  function(pkt, keys, shapes, positions, rotations) {
    var projectiles = pkt.keysMatching(
      'point-shape', 'verlet-position',
      'rotation', 'projectile');
    // Actual collision code
  }
)
```

This label is the only "asteroidish" thing about this system

```
pkt.system(
  'asteroid-projectile-collider',
  ['point-shape', 'verlet-position',
   'rotation', 'ship'],
  function(pkt, keys, shapes, positions, rotations) {
    var projectiles = pkt.keysMatching(
      'point-shape', 'verlet-position',
      'rotation', 'projectile');
    // Actual collision code
  }
)
```

THE HUNTER
HAS BECOME
THE HUNTED

There is still more work to do!

But not today.

ProCon: Professional Conferences for Professionals

- Logic is spread out
- State is concentrated in components
- Easier to optimize for batch drawing, batch collisions, etc.
- Extremely modular

I mentioned databases a long time ago.

- ComponentTypes — Tables (properties are columns)
- Components / Component Data — Rows
- Keys — Primary Keys
- Systems — Stored Procedures? (not really)

- ComponentTypes — Tables (properties are columns)
- Components / Component Data — Rows
- Keys — Primary Keys
- ~~Systems — Stored Procedures? (not really)~~

- ComponentTypes — Tables (properties are columns)
- Components / Component Data — Rows
- Keys — Primary Keys
- Systems — Queries + Business Logic

```
console.log(JSON.stringify(pkt.components))
```

```
{
  "ctx-2d": {
    "1": {
      "cvs": {},
      "ctx": {},
      "center": {
        "x": 719.5,
        "y": 200.5
      },
      "width": 1439,
      "height": 401
    }
  },
  "game-config": {
```

```
},
"verlet-position": {
  "4": {
    "cpos": {
      "x": 719.5,
      "y": 397.5
    },
    "ppos": {
      "x": 719.5,
      "y": 397.5
    },
    "acel": {
      "x": 0,
      "y": 0
    }
```

```
    }
  }
},
"rotation": {
  "4": {
    "angle": 0,
    "rate": 0.1
  },
  "5": {
    "angle": 0,
    "rate": 0.1
  },
  "6": {
    "angle": 0,
    "rate": 0.1
```

“But did you finish the game jam?”

–Everyone?


Prepare for
IMMEDIATE
Truth

Sorry.

There was no jam.

It was a conceit of the talk.

I hope that's alright.

Thank You!

@KirbySaysHi

http://github.com/kirbysayshi/pocket-ces

Thank You!

@KirbySaysHi

http://github.com/kirbysayshi/pocket-ces