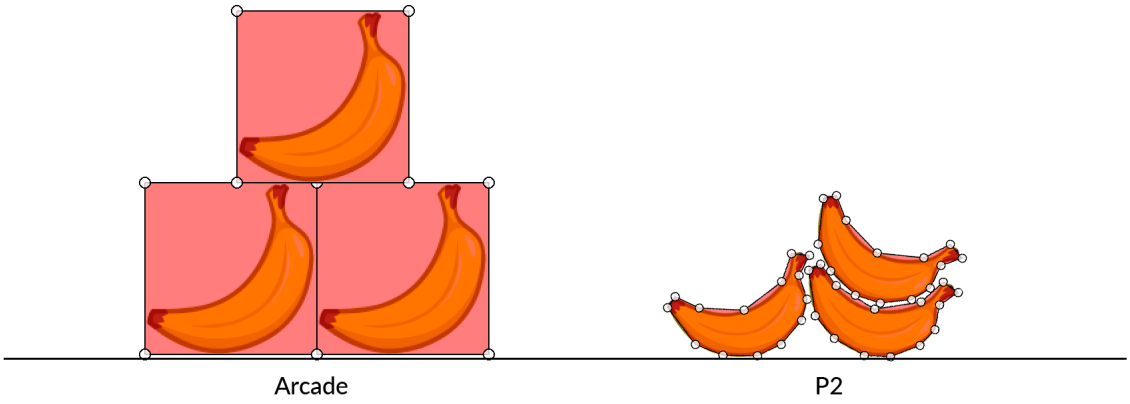
Week 9: All About Arcade Physics in Phaser.js

Working with the Arcade Physics System in Phaser

In last week's materials, we continued our tour of many of basics of the Phase game engine. In this week, we focus on the collection of features that the game engine offers us to simulate the physical world and the natural behavior of objects in that world, where gravity, friction and bounciness exist to name just a few.

Physics Systems: Arcade, P2, and Ninja

Phaser supports at least three different physics systems by default: Arcade, P2, and Ninja physics. (Box2D is also supported by a commercial plugin). We will leave a discussion of the Ninja system for another time. Arcade physics is the simpler and less-precise system for working quickly with sprite physics in a game. The P2 physics system offers extremely precise physics simulations in comparison to arcade. For instance, the hit box, body, or active collision target area for a sprite in the arcade physics system is always a rectangular bounding box. While the P2 physics system allows you to define multi-segment polygons for each sprite's hit box body. The illustration below demonstrates the usefulness of such precise physics modeling. Notice the banana sprites on the right, in P2 physics, can rest comfortably nestled into each of their neighbor's complex hitbox outlines. On the left, those arcade physics banana sprites get left stacked as though each was in its own invisible box – sometimes called the sprite's bounding box.



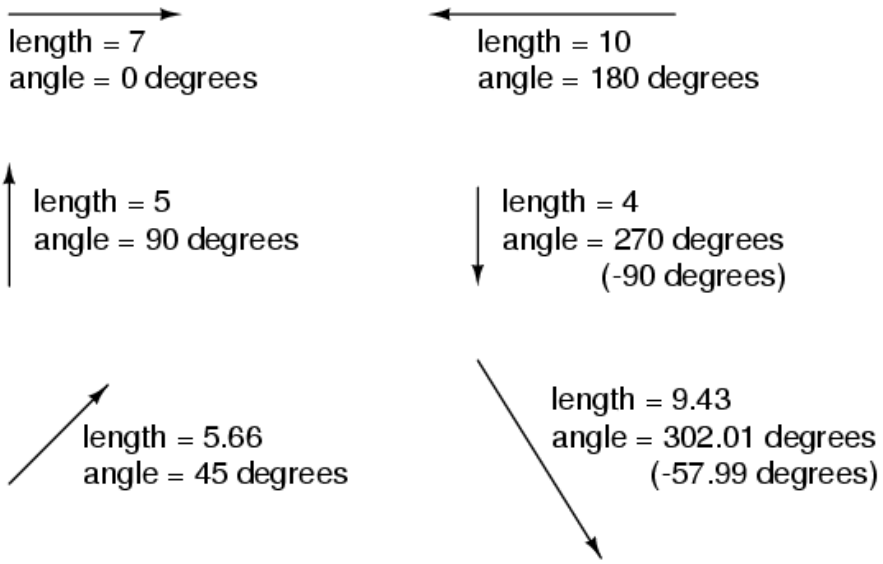
Concepts of Game Engine Physics Systems

Simulation of the physical world is an important capability of any game engine. The physics systems in game engines utilize vector mathematics and linear algebra. Newton’s motion equations let us consider

overall sprite motion in smaller equations, one each for the x and y dimensions.

This means when we simulate a ball in 2D, we can solve two separate equations for how it moves vertically and horizontally. When you throw a ball, it moves forward at a constant velocity. It also moves vertically at the same time. Its vertical motion is anything but constant, because it is affected by gravity.

So while we could model this motion with two separate equations, why should we when linear algebra can aid us with the concept of vectors. Vectors are like arrows: they have a length and an angle (or direction). Vectors simplify many of the common tasks that physics systems must accomplish. Using vectors also allows operations like adding and subtracting vectors, and points can be modeled as vectors.



Newton's Second Law of Motion

Newton's Second Law of motion relates an object's acceleration to the amount of force applied to that object. The formula itself is:

```
F = ma
```

The "F" and "a" here represent vectors. This means that, if you wanted to, you could split this equation up into three parts and use each separately:

```
Fx = max
Fy = may
Fz = maz
```

Newton's Second law is invaluable to physics systems. It will guide almost every one of our object interactions. This formula is so powerful because:

1. forces are how objects in the real world interact with each other
2. if you know an object's acceleration, you can calculate both its velocity and its position.

Once the physics system can model the forces correctly, it can model nearly any physical scenario with some accuracy. The process for simulating an sprite's motion goes something like this:

1. Figure out what the forces are on an sprite
2. Add those forces up to get a single "resultant" or "net" force
3. Use F = ma to calculate the sprite's acceleration due to those forces
4. Use the sprite's acceleration to calculate the sprite's velocity
5. Use the sprite's velocity to calculate the sprite's position
6. Since the forces on the sprite may change from moment to moment, repeat this process from #1, forever.

So forces are where it is all at in this model. We next look at some of these forces in detail.

This topic is discussed in textbook section 6.15.

Bodies

Note that a sprite in Phaser will not have a physics system applied to it unless the *body* property is utilized.

The physics **body** property is available for each sprite. All physics operations must be performed against the **body** rather than the sprite itself. You set the velocity, acceleration, and bounce values all on the **body**.

This topic is discussed in textbook section 6.17.

Velocity

The speed that a sprite's body (its physical hitbox) moves can also be called its velocity. In the 2D world of Phaser, velocity has horizontal and vertical properties: **x** and **y**. Phaser's arcade physics system also support angular velocity

```
sprite.body.velocity.x = 0;
sprite.body.velocity.y = 0;
sprite.body.angularVelocity = 0;
```

This topic is discussed in textbook section 6.15.1.

Acceleration Forces

The acceleration is the rate of change of the velocity. It is measured in pixels per second squared in Phaser. Angular acceleration is also supported. Maximum velocity and maximum angular velocity can be set.

```
sprite.body.maxVelocity.set(200);
sprite.body.maxAngular = 500;
sprite.body.acceleration.set(0);
```

This topic is discussed in textbook section 6.15.2.1.

Friction Forces

Phaser supports the **drag** property for sprites to model the resistance to the sprite's movement.

```
sprite.body.drag.x = 5;
```

This topic is discussed in textbook section 6.15.2.2.

Bounciness (or Restitution Forces)

Phaser supports the bounce property for sprites to model the amount of energy transfered from the bounce.

```
sprite.body.bounce.set(0.8);
```

This topic is discussed in textbook section 6.15.2.3.

Collisions

The intersection of two sprites and/or groups is managed differently by each physics system in Phaser. The simpler arcade physics system can only detect intersections based on a sprite's bounding box. P2 and Ninja physics systems offer more sophisticated collision detection.

This topic is discussed in textbook section 6.15.2.4.

Arcade Physics System

The arcade physics system is easiest to work with since you do not have to define hitbox body shapes for sprites. The bounding box is always the hitbox for sprites in arcade physics. Arcade physics uses the concept of quad trees to rapidly analyze groups of sprites based on their placement within quadrants of the screen. This quad tree technique also Phaser to render frames more quickly. Arcade physics supports a simple gravity property for each sprite's body that can assign a force to the sprite. Objects can also be immovable and ignore the force of gravity. Each sprite can have drag applied to affect the velocity of the sprite. Angular velocity and bounce properties are also available for a sprite's body in arcade physics.

This topic is discussed in textbook section 6.18.

P2 Physics System

The P2 physics system offers additional factors and forces to better simulate the real world physics of sprite behavior. Each sprite may have a complex shape-based hitbox or body defined. Software applications exist to provide tools for authoring polygon and circle based sprite shapes. **Loon Physics (https://loonride.com/physics)** and **PhysicsEditor (https://www.codeandweb.com/physicseditor)** are two available editors. Unlike arcade physics, the P2 physics system processes collisions between sprites

automatically. This means forces and reactions are calculated and updated property values are applied to the sprites involved in any collision. You can also assign additional handlers to respond to sprite collisions. A contact signal is thrown whenever sprites touch. This signal can be used to trigger your own custom code. The P2 system also supports constraints like lock, distance and spring, allowing multiple sprites to be related in a systemic model so if one sprite moves, the others in the constraint system respond. Rounding out the modeling of the physical world, P2 supports modeling of contact materials. With this distinction, material surface characteristics can be simulated. Slippery, rough and bouncy surfaces can be modeled.

This topic is discussed in textbook section 6.19.