Making Physics Bodies Move

Move a body using various physics properties, like velocity, gravity or impulses.

Overview

By default, only gravity is applied to physics bodies in the scene. In some cases, that might be enough to build a game. But in most cases, you need to take other steps to change the speed of physics bodies.

First, you can control a physics body's velocity directly, by setting its `velocity` and `angularVelocity`properties. As with many other properties, you often set these properties once when the physics body is first created and then let the physics simulation adjust them as necessary. For example, assume for a moment you are making a space-based game where a rocket ship can fire missiles. When the ship fires a missile, the missile should have a starting velocity of the ship plus an additional vector in the direction of the launch. The following code shows one implementation for calculating the launch velocity.

LISTING 1 CALCULATING THE MISSILE'S INITIAL VELOCITY

```
missile.physicsBody?.velocity = self.physicsBody!.velocity
missile.physicsBody?.applyImpulse(CGVector(dx: missileLaunchImpulse *
cos(shipDirection),
                                           dy: missileLaunchImpulse *
sin(shipDirection)))
```

When a body is in the simulation, it is more common for the velocity to be adjusted based on forces applied to the body. Another source of velocity changes, collisions, is discussed later.

The default collection of forces that apply to a body include:

>> The gravitational force applied by the physics world

>> The damping forces applied by the body's own properties

>> A frictional force based on contact with another body in the system

You can also apply your own forces and impulses to physics bodies. Most often, you apply forces and impulses in a pre-processing step before the simulation executes. Your game logic is responsible for determining which forces need to be applied and for making the appropriate method calls to apply those forces.

You can choose to apply either a force or an impulse:

>> A force is applied for a length of time based on the amount of simulation time that passes between when you apply the force and when the next frame of the simulation is processed. So, to apply a continuous force to an body, you need to make the appropriate method calls each time a new frame is processed. Forces are usually used for continuous effects

>> An impulse makes an instantaneous change to the body's velocity that is independent of the amount of simulation time that has passed. Impulses are usually used for immediate changes to a body's velocity.

To continue with the rocket example, a rocket ship probably applies a force to itself when it turns on its engines. However, when it fires a missile, it might launch the missile with the rocket's own velocity and then apply a single impulse to it to give it the initial burst of speed.

Because forces and impulses are modeling the same concept—adjusting a body's velocity—the remainder of this section focuses on forces.

You can apply a force to a body in one of three ways:

>> A linear force that only affects the body's linear velocity.

>> An angular force that only affects the body's angular velocity.

>> A force applied to a point on the body. The physics simulation calculates separate changes to the body's angular and linear velocity, based on the shape of the object and the point where the force was applied.

The following code shows code you could implement in a sprite subclass to apply a force to the ship. This force accelerates the rocket when the main engines are activated. Because the engines are at the back of the rocket, the force is applied to linearly to the rocket body. The code calculates the thrust vector based on the current orientation of the rocket. The orientation is based on the `zRotation` property of the corresponding node, but the orientation of the artwork may differ from the orientation of the node. The thrust should always be oriented with the artwork. The following code shows a similar effect, but this time the rocket is being rotated by the force, so the thrust is applied as an angular thrust.

LISTING 2 APPLYING LATERAL THRUST

```
self.physicsBody?.applyTorque(thrust)
```