Phaser HTML5 Game Tutorial: Build A Pong Game 3: Moving the Ball

June 17, 2015 By Leave a Comment

In part 1[1] of this tutorial series, we set up our project to build a Pong game.

In part 2[2], we loaded and added some sprites to our game.

Now, we'll work on getting our ball to move around. Like the original 1972 game, we'll start by creating a demo mode where the ball will be bouncing within the stage area.

Start by adding two functions: `initPhysics`, `startDemo` and `startBall` after the `initGraphics` function like what you see here:

paddleRightSpritespritegamePropertiespaddleRight_xworldcenterYgraphicAssetspaddleName

paddleRightSpriteanchor

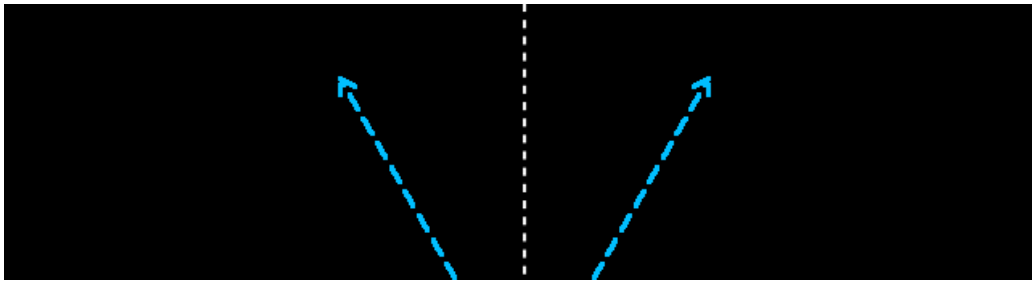
initPhysicsfunction

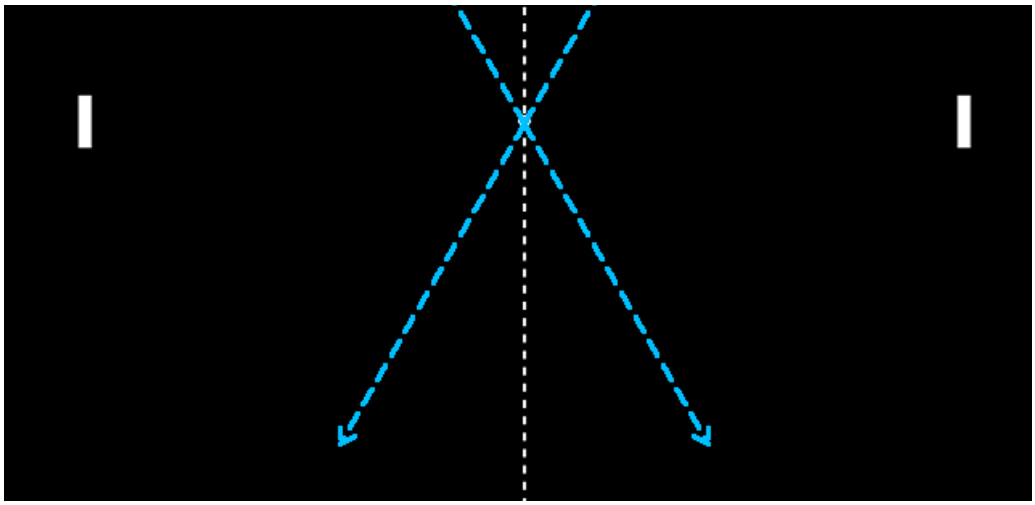
startDemofunction

startBallfunction

The `initPhysics` function is used to enable the Phaser physics system and add physics bodies to all the game objects. The `startDemo` function will be used to run our demo state while the `startBall` function will be used to get our ball moving.

When the ball first appears on screen and starts moving, we will set it to move at a fixed velocity and randomly choose one of four possible directions. The following illustration shows what we're going to achieve:





Going back to the `gameProperties` object, we'll add the following code:

```
gameProperties

screenWidth

screenHeight

dashSize

paddleLeft_x

paddleRight_x

ballVelocity

 ballStartDelay

ballRandomStartingAngleLeft

ballRandomStartingAngleRight
```

The `ballVelocity` property sets the speed of the ball. The value given is measured in pixels per second so the ball will move at a rate of 500 pixels per second.

The `ballStartDelay` property will be used to delay the ball for 2 seconds before it starts moving. This will be useful later on as well when we need to reset the ball after a player has scored a point.

The `ballRandomStartingAngleLeft` property provides the starting angles in degrees when the ball is supposed to move towards the left while the `ballRandomStartingAngleRight` property is used when the ball moves towards the right. We will join these two arrays in the `startBall` function so our ball has 4 random starting directions

to choose from.

We'll be using arcade physics system to check for sprite collisions, overlap and motion. Phaser also includes the P2 and Ninja physics system that will only be used if we need more accurate and complex physics collisions. In this case, we use the arcade physics system as it's is a high performance system that only checks when two sprites overlap. It is good enough for use in this tutorial.

In the `initPhysics` function, let's add the following code:

initPhysicsfunction

physicsstartSystemPhaserPhysicsARCADE

physicsenableballSprite

ballSpritecheckWorldBounds

ballSpritecollideWorldBounds

ballSpriteimmovable

ballSpritebounce

The Phaser framework only allows one physics system to be in use at this time.

In line 86, we choose the arcade physics system and create an instance of the requested physics system.

In line 87, we create the default arcade physics body on the `ballSprite` object.

Next, in live 89, we set the `checkWorldBounds` to `true`. This will cause the `ballSprite` object to dispatch an event when it leaves the game world. We'll use this later on to check which side scores a point.

We also want the `ballSprite` to rebound back into the game world when it collides with the world boundaries so in line 90, we set the physics `body.collideWorldBounds` property to `true`.

To prevent the `ballSprite` object from receiving any forces when colliding with the paddles, we set the `body.immovable` property to `true`.

Lastly, in line 92, we set the `body.bounce` property to 1. This ensures that the velocity of the `ballSprite` remains the same when it collides with any object. This property works like a velocity multiplier when a physics object collides with another. For example, if we set it to 0.5 and the ball is travelling at 500 velocity, the ball's velocity will reduce to 250 when it collides with another physics body.

Now we'll add some code to the `startDemo` function:

startDemofunction

ballSpritevisiblefalse

eventsPhaserTimerSECONDgamePropertiesballStartDelaystartBall

First, we hide our ball sprite by setting it's `visible` property to `false`. Next, we'll add a `Timer` to start our ball moving after a few seconds. The `Timer` creates an object that waits for a specified moment in time to run a specified callback function. A `Timer` uses milliseconds as its unit of time. Note that there are 1000 milliseconds in 1 second. So in this case, a delay to 2000 would fire the event in 2 seconds.

We use 3 arguments when adding this `Timer` event:

`delay`: The number of milliseconds that should pass before the callback function is run.
`callback`: The callback function that will be run when the Timer event occurs.
`callbackContext`: The context or scope where the callback will be called.

For the `delay`, we call the `Phaser.Timer.SECOND` which gives us the value of 1000. Multiply 1000 by the `gameProperties.ballStartDelay`, we have a total of 2000 milliseconds or 2 seconds. For the callback argument we will be calling the `startBall` function. Notice we use the keyword to refer to the mainState object.

Finally, let's add the following code to the `startBall` function:

startBallfunction

ballSpritevisible

randomAnglegamePropertiesballRandomStartingAngleRightconcatgamePropertiesballRandomStartingAngleLeft

physicsarcadevelocityFromAnglerandomAnglegamePropertiesballVelocityballSpritevelocity

The ball sprite was hidden previously in the `startDemo` function by setting it's `visible` property to `false`. Here we make it visible by setting it to `true`.

Next we pick a random angle from the `ballRandomStartingAngleRight` array and the `ballRandomStartingAngleLeft` array. The `game.rnd.pick` function is used to pick a random item from an array. While the `concat` function is used to join the `ballRandomStartingAngleRight` and `ballRandomStartingAngleLeft` arrays.

The last thing we need to for this step is set our ball velocity to get it moving. Here we use the `game.physics.arcade.velocityFromAngle` function that calculates the x-velocity (horizontal) and y-velocity (vertical) for the ball based on the `randomAngle` and the `gameProperties.ballVelocity` values.

Here's what your game should look something like:

Download the source files for this step here[3].

In step 4[4] of this tutorial, we'll add the game play state and controls to move our paddles.

1. http://zekechan.net/getting-started-html5-game-development-pong1/
2. http://zekechan.net/getting-started-html5-game-development-pong2/
3. https://github.com/zekechan/phaser-html5-tutorial-pong/releases/download/1.0/3-Making_the_ball_move.zip
4. http://zekechan.net/getting-started-html5-game-development-pong4/