# Tutorial – Design an Asteroids Game

As promised last week, this is where we start to do the cool stuff.

We'll start by setting up a game loop for our game. We'll then cover drawing images so we can get our player ship drawing to the screen, and we'll even start moving it around the screen by handling the keyboard key press events. While we're at it, we'll also cover how to rotate the ship and move it so no matter what direction it's facing, it's always moving forward.

By the end of this lesson you'll have a basic game loop set up with a player character that you can move around the screen via the keyboard.

For this tutorial we'll need the code from last week's tutorial. The HTML file will remain the same, so we're only going to modify the main.js file.

## Adding the Game Loop:

1. We don't need much from last week. The only code we'll need is the two lines up the top that get the canvas and the 2D context. So let's start off with a main.js file that looks like this:

```
var canvas = document.getElementById("gameCanvas");
var context = canvas.getContext("2d");
```

   You can delete everything else.

2. We'll start by adding the run() function, and the chunk of code that will make sure the run() function is called at up to 60 frames per second.

   This code is a little advanced, so it's not important that you understand 100% of what is going on here.

   Put this code at the very end of the file. This same piece of code will be in every program we write from now, and it should always come at the end of the file.

```
function run()
{
}


//-------------------- Don't modify anything below here


// This code will set up the framework so that the 'run' function is
// called 60 times per second. We have some options to fall back on
// in case the browser doesn't support our preferred method.
(function() {
  var onEachFrame;
  if (window.requestAnimationFrame) {
    onEachFrame = function(cb) {
      var _cb = function() { cb(); window.requestAnimationFrame(_cb); }
      _cb();
    };
  } else if (window.mozRequestAnimationFrame) {
    onEachFrame = function(cb) {
      var _cb = function() { cb();
window.mozRequestAnimationFrame(_cb); }
      _cb();
    };
  } else {
    onEachFrame = function(cb) {
      setInterval(cb, 1000 / 60);
    }
  }

  window.onEachFrame = onEachFrame;
})();

window.onEachFrame(run);
```

Basically, we are calling a function of the browser window that lets us request to execute the run() function at 60 frames per second… if it can. This is only a request, so the frame rate might vary.

We have a bit of code in there that handle some browser specific situations, because all the browsers implement HTML5 a little differently. On Firefox (Mozilla) we need to use a slightly different request function.

We also have a fallback that will use a slightly older method of calling the run() function, just in case the requestAnimationFrame() function isn't available.

## Drawing the Player:

Our game loop is set up but the run() function isn't drawing anything yet, so you won't see anything drawing to the screen if you try to execute the program.

Let's start off by clearing the screen, and then draw the ship sprite that will be our player.

1. To start with, we'll set up the game to clear the screen every frame. It's a good idea to clear the screen every time we draw a frame, otherwise you'll see the previous frame.

   To clear the screen all we need to do is draw a filled rectangle that is the same size as the screen. I'll fill mine with a grey colour so that the canvas is obvious, but you could use any colour you like.

   Add the following code to your run() function:

```
function run()
{
	context.fillStyle = "#ccc";
	context.fillRect(0, 0, canvas.width, canvas.height);
}
```

   If you want to use a different colour, modify the fillStyle variable. A list of common colours is available from this site: http://www.w3schools.com/tags/ref_colornames.asp

2. Awesome. We now have a grey square drawing at up to 60 fps.

   Let's beef things up a bit and draw the player's plane. If you don't have a sprite that you'd like to use you can grab the plane sprite for this tutorial.

   Modify the program so it looks like this:

```
var player = document.createElement("img");
player.src = "ship.png";

var x = 10;
var y = 10;

function run()
{
	context.fillStyle = "#ccc";
	context.fillRect(0, 0, canvas.width, canvas.height);

	context.drawImage(player, x, y);
}
```

This code loads the ship image and creates an x and a y variable to store the position of the ship image (which is called player). We then draw the ship at the position indicated by the x and y variables.

Make sure your ship image is in the same directory as your main.js file, otherwise it won't load and you won't see it drawing.

## Adding Keyboard Events:

While it may be pretty to look at, our game doesn't do much at the moment.

We want to be able to move our ship around the screen using the arrow keys on our keyboard. We'll add some code to do that now.

1. Just **before** the declaration of the player variable ('var player') add the following code:

```
    // set up some event handlers to process keyboard input
window.addEventListener('keydown', function(evt) { onKeyDown(evt); }, false);
window.addEventListener('keyup', function(evt) { onKeyUp(evt); }, false);
```

This sets up some event handlers that will fire whenever the keydown or keyup events occur.

2. When a key is pressed on the keyboard the onKeyDown function will execute. Likewise when the key is released the onKeyUp function will be called. We'll write the onKeyDown and onKeyUp functions now. You can put these functions just before the run() function.

```
function onKeyDown(event)
{
	if(event.keyCode == KEY_UP)
	{
		y += 1;
	}
	if(event.keyCode == KEY_DOWN)
	{
		y -= 1;
	}

	if(event.keyCode == KEY_LEFT)
	{
		angle -= 0.02;
	}
	if(event.keyCode == KEY_RIGHT)
	{
		angle += 0.02;
	}
}
```

```
function onKeyUp(event)
{
}
```

3. You may have noticed that we snuck in a new variable called angle. We'll need to create that now.

   Declare the angle variable just under where you declared the x and y variables:

```
var x = 10;
var y = 10;
var angle = 0;
```

4. Some of you may have noticed that in the onKeyDown function where comparing the event.keyCode to values like KEY_LEFT or KEY_RIGHT, but in the lesson slide we used a value like 38.

   In fact, if you tried to run this program now it wouldn't work, and you'd see in the Chrome console window the message that KEY_UP is not defined.

   The KEY_UP that Chrome is referring to, and that we're using here, is actually a variable.

   So we can read our code easier – and so there are no magic numbers – we'll store our key codes in some 'constant' values. By making the variable names all upper-case, we'll be able to remember to never change the value of these variables.

   Insert the following code somewhere **before** the onKeyDown() function.

```
    // key constants
    // Go here for a list of key codes:
    // https://developer.mozilla.org/en-US/docs/DOM/KeyboardEvent
var KEY_SPACE = 32;
var KEY_LEFT = 37;
var KEY_UP = 38;
var KEY_RIGHT = 39;
var KEY_DOWN = 40;
```

5. If you run the program now you should be able to make your ship go forwards and backwards using the up and down arrow keys on your keyboard. Awesome.

6. The last thing we need to add is rotation using the angle variable we created.

   Rotation is actually a little tricky, and I don't expect you to get it on your own. Update the run() function so that it looks like this:

```
function run()
{
	context.fillStyle = "#ccc";
	context.fillRect(0, 0, canvas.width, canvas.height);

	context.save();
		context.translate(x, y);
		context.rotate(angle);
		context.drawImage(player, -player.width/2, -player.height/2);
	context.restore();
}
```

This code doesn't actually rotate the image, instead it rotates the canvas, draws the image, then restores the canvas back to normal. The result is that the image appears to be rotated.

However it still doesn't look right. Instead of going forward and backwards in the direction it's currently facing, the ship just moves up and down the screen.

To solve this takes a bit of math. This is also a little trick, so everything you'll need will be included here.

## Moving the Ship (with rotation):

To move the ship based on its current rotation (so when we press the up key it moves forward and not just up the screen), we need to do a bit of math.

Basically when we increment or decrement the y axis during the onKeyDown function, we'll instead need to calculate how much x movement and how much y movement is needed based on the current angle.

This isn't hard, but it's not obvious.

Update the onKeyDown() function as follows:

```
function onKeyDown(event)
{
	if(event.keyCode == KEY_UP)
	{
		// figure out the x and y movement based on the current rotation
		// (so that if the ship is moving forward, then it will move
		// forward according to its rotation. Without this, the ship
		// would just move up the screen when we pressed 'up',
		// regardless of which way it was rotated)
		// for an explanation of this formula, see
		// http://en.wikipedia.org/wiki/Rotation_matrix
		// calculate sin and cos for the player's current rotation
		var s = Math.sin(angle);
		var c = Math.cos(angle);
		var xSpeed = 0;
		var ySpeed = 1;
		x += (xSpeed * c) - (ySpeed * s);
		y += (xSpeed * s) + (ySpeed * c);
	}
	if(event.keyCode == KEY_DOWN)
	{
		var s = Math.sin(angle);
		var c = Math.cos(angle);
		var xSpeed = 0;
		var ySpeed = -1;
		x += (xSpeed * c) - (ySpeed * s);
		y += (xSpeed * s) + (ySpeed * c);
	}

	if(event.keyCode == KEY_LEFT)
	{
		angle -= 0.02;
	}
	if(event.keyCode == KEY_RIGHT)
	{
		angle += 0.02;
	}
}
```

I've made variables for the x-axis speed and y-axis speed so you can see the complete formula, but in reality these will always be 0 (no left or right movement) or 1/-1 (forwards and backwards movement).

The formula is a simplification of the rotation matrices used in linear algebra. This kind of matrix math is common in 2D and 3D game development and is covered in the advanced diploma course, but is outside the scope of this course.

## Exercise:

Congratulations, you should now have a game that uses a game loop, and can respond to keyboard events to move and rotate a ship on the screen.

We're getting closer to a real asteroids game now.

As an exercise see if you can modify the player's movement and rotation speeds.

Can you add the ability for the player to strafe left or right if the 'A' or 'S' key is pressed?