Tutorial – Arrays in JavaScript

In this tutorial we'll be continuing from where we left off in the last tutorial.

We'll start by adding a tiled background. Then we'll update our game so that one asteroid will spawn every second. Finally, we'll finish up by allowing the player to shoot more than one bullet.

These features require the use of loops and arrays, so make sure you understand this week's lecture.

Adding Delta Time:

Timing is important for your games. Now that we know what delta time is, and why it's important, add the getDeltaTime() function from the lecture slides to your game. Put it right up the top of your main.js file, just under the window.addEventListener functions.

(In case you missed it, the code is on the last slide for this week's lecture.)

Adding a Tiled Background:

1. Prepare an image that is 32 pixels wide by 32 pixels high, or use the sample image provided. This will be the image we will use to draw the tiled background.
2. Before we can do anything, we need to load the image that will be used to draw the background.

   Add the following code somewhere near the top of your program:

```
    // load the image to use for the tiled background
var grass = document.createElement("img");
grass.src = "grass.png";
```

3. We are going to create a 2D array to store the image to use when drawing each tile.

   Our screen is 640 x 480 pixels, and our tile is 32 x 32 pixels, so we will need a grid that is 20 columns wide by 15 columns high.
   Each cell in our grid will store the image to draw for that tile.

   Later, if you wanted to, you could experiment with putting some different images in various cells of your grid.

Let's create our 2D array (our grid of tiles), and fill it with our image data. Insert this code just below where you loaded the grass image:

```
	// create the tiled background
var background = [];

for(var y=0;y<15;y++)
{
	background[y] = [];
	for(var x=0; x<20; x++)
		background[y][x] = grass;
}
```

4. And finally, add the following code to the start of the run() function:

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

	var deltaTime = getDeltaTime();

		// draw the tiled background (make sure you do this first, so everything
		// else in the scene will be drawn on top of the tiled background)
	for(var y=0; y<15; y++)
	{
		for(var x=0; x<20; x++)
		{
			context.drawImage(background[y][x], x*32, y*32);
		}
	}
```

Remember: we add this to the start of the function so that the tiles will be drawn below everything else.

Spawning Asteroids:

At the moment we only have one asteroid. That's not very interesting. Now that we know a bit about arrays, we'll create an asteroid array and set up our game to spawn a new asteroid every second.

When an asteroid spawns, we'll set its direction so that it moves towards the player's position at the time it spawned.

1. Remove the code that creates the asteroid variable and replace it with the following code

```
	// this creates the asteroid object and assigns it some properties
var asteroid = {
	image: document.createElement("img"),
	x: 100,
	y: 100,
	width: 69,
	height: 75,
	directionX: 0,
	directionY: 0,
	isDead: false
};
	// set the image for the asteroid to use
asteroid.image.src = "rock_large.png";

asteroid.directionX = 10;
asteroid.directionY = 8;
	// 'normalize' the velocity (the hypotenuse of the triangle formed by x,y
	// will equal 1)
var magnitude = Math.sqrt( (asteroid.directionX * asteroid.directionX) +
                  (asteroid.directionY * asteroid.directionY) );
if(magnitude != 0)
{
	asteroid.directionX /= magnitude;
	asteroid.directionY /= magnitude;
}

	// Create an array to store our asteroids
var asteroids = [];
```

2. Now we need some code to spawn a new asteroid. We're going to create a new function, and we'll call this function once every second during the run() function.

```
// rand(floor, ceil)
// Return a random number within the range of the two input variables
function rand(floor, ceil)
{
	return Math.floor( (Math.random()* (ceil-floor)) +floor );
}

// Create a new random asteroid and add it to our asteroids array.
// We'll give the asteroid a random position (just off screen) and
// set it moving towards the center of the screen
function spawnAsteroid()
{
	// make a random variable to specify which asteroid image to use
	// (small, mediam or large)
	var type = rand(0, 3);

	// create the new asteroid
	var asteroid = {};

	asteroid.image = document.createElement("img");
	asteroid.image.src = "rock_large.png";
	asteroid.width = 69;
	asteroid.height = 75;
	// to set a random position just off screen, we'll start at the centre of the
	// screen then move in a random direction by the width of the screen
	var x = SCREEN_WIDTH/2;
	var y = SCREEN_HEIGHT/2;

	var dirX = rand(-10,10);
	var dirY = rand(-10,10);
	// 'normalize' the direction (the hypotenuse of the triangle formed
	// by x,y will equal 1)
	var magnitude = (dirX * dirX) + (dirY * dirY);
	if(magnitude != 0)
	{
		var oneOverMag = 1 / Math.sqrt(magnitude);
		dirX *= oneOverMag;
		dirY *= oneOverMag;
	}
	// now we can multiply the dirX/Y by the screen width to move that amount from
	// the centre of the screen
	var movX = dirX * SCREEN_WIDTH;
	var movY = dirY * SCREEN_HEIGHT;

	// add the direction to the original position to get the starting position of the
	// asteroid
	asteroid.x = x + movX;
	asteroid.y = y + movY;

	// now, the easy way to set the velocity so that the asteroid moves towards the
	// centre of the screen is to just reverse the direction we found earlier
	asteroid.velocityX = -dirX * ASTEROID_SPEED;
	asteroid.velocityY = -dirY * ASTEROID_SPEED;

	// finally we can add our new asteroid to the end of our asteroids array
	asteroids.push(asteroid);
}
```

3. Finally, update the run() function so that we spawn an asteroid every second and add it to the asteroids array. While we're here, we'll also modify the existing asteroid code so that we update all asteroids in the asteroids array.

```
var spawnTimer = 0;
function run()
{
        . . .
        // update and draw the bullet
        // we want to do this first so that the plane is drawn on top of the bullet
        if(bullet.isDead == false)
        {
                bullet.x += bullet.velocityX;
                bullet.y += bullet.velocityY;
                context.drawImage(bullet.image,
                        bullet.x - bullet.width/2, bullet.y - bullet.height/2);
                if(asteroid.isDead == false)
                {
                        var hit = intersects(
                          bullet.x - bullet.width/2, bullet.y - bullet.height/2,
                          bullet.width, bullet.height,
                          asteroid.x - asteroid.width/2, asteroid.y - asteroid.height/2,
                          asteroid.width, asteroid.height);
                        if(hit == true)
                        {
                                bullet.isDead = true;
                                asteroid.isDead = true;
                        }
                }
                if(bullet.x < 0 || bullet.x > SCREEN_WIDTH ||
                        bullet.y < 0 || bullet.y > SCREEN_HEIGHT)
                {
                        bullet.isDead = true;
                }
        }
        . . .

        // update all the asterioids in the asteroids array
        for(var i=0; i<asteroids.length; i++)
        {
                // update the asteroids position according to its current velocity.
                // TODO: Dont forget to multiply by deltaTime to get a constant speed
                asteroids[i].x = asteroids[i].x + asteroids[i].velocityX;
                asteroids[i].y = asteroids[i].y + asteroids[i].velocityY;

                // TODO: check if the asteroid has gone out of the screen boundaries
                // If so, wrap the astroid around the screen so it comes back from the
                // other side
        }

        // draw all the asteroids
        for(var i=0; i<asteroids.length; i++)
        {
                context.drawImage(asteroids[i].image, asteroids[i].x - asteroids[i].width/2,
                                asteroids[i].y - asteroids[i].height/2);
        }
        spawnTimer -= deltaTime;
        if(spawnTimer <= 0)
        {
                spawnTimer = 1;
                spawnAsteroid();
        }
}
```

Because we've got multiple asteroids now, we need to delete the code that tests for collisions between the bullet and the old asteroid. We'll add new collision code soon, but for now make these changes and confirm that your program runs and a new asteroid is spawning every second.

Adding More Bullets:

Now that we have multiple asteroids, we'll need to be able to shoot more than one bullet at a time.

The process will be similar to that for the asteroids. We'll remove our single bullet and create a bullet list instead. We'll then update our shoot function to create a new bullet and add it to the list when the player fires. To make sure we can't spam the fire button, we'll also add a cool-down timer.

1. Remove the code to create a single bullet and replace it with code to create a new empty array called 'bullets':

```
~~	// this creates the bullet object and assigns it some properties~~
~~var bullet = {~~
~~	image: document.createElement("img"),~~
~~	x: player.x,~~
~~	y: player.y,~~
~~	width: 5,~~
~~	height: 5,~~
~~	velocityX: 0,~~
~~	velocityY: 0,~~
~~	isDead: true~~
~~};~~
~~	// set the image for the asteroid to use~~
~~bullet.image.src = "bullet.png";~~

	// create all the bullets in our game
var bullets = [];
```

2. Update the playerShoot() function so that we create a new bullet object and add it to the bullets list whenever the player fires:

```
function playerShoot()
{
	var bullet = {
		image: document.createElement("img"),
		x: player.x,
		y: player.y,
		width: 5,
		height: 5,
		velocityX: 0,
		velocityY: 0
	};

	bullet.image.src = "bullet.png";

	// start off with a velocity that shoots the bullet straight up
	var velX = 0;
	var velY = 1;

	// now rotate this vector acording to the ship's current rotation
	var s = Math.sin(player.rotation);
	var c = Math.cos(player.rotation);

	// for an explanation of this formula,
	// see http://en.wikipedia.org/wiki/Rotation_matrix
	var xVel = (velX * c) - (velY * s);
	var yVel = (velX * s) + (velY * c);

	bullet.velocityX = xVel * BULLET_SPEED;
	bullet.velocityY = yVel * BULLET_SPEED;

	// finally, add the bullet to the bullets array
	bullets.push(bullet);
}
```

Note we no longer need the bullet.isDead variable (if a bullet dies, we'll just remove it from the bullets array).

3. We want to change the shooting function so that the player can hold down fire and keep firing. We also want to add a cool-down timer so that the player can't fire too many bullets at once.

   To do this, the first thing we need to do is remove the call to the playerShoot() function from the onKeyUp() function:

```
function onKeyUp(event)
{
	. . .

	if(event.keyCode == KEY_SPACE)
	{
		playerShoot();
	}
}
```

4. Modify the onKeyDown() function as follows:

```
var shootTimer = 0;
function onKeyDown(event)
{
	. . .

	if(event.keyCode == KEY_SPACE && shootTimer <= 0)
	{
		shootTimer += 0.3;
		playerShoot();
	}
}
```

5. Update the run() function as follows:

```
function run()
{
        . . .

        // update and draw the bullet we want to do this first
        // so that the plane is drawn on top of the bullet
        if(bullet.isDead == false)
        {
                bullet.x += bullet.velocityX;
                bullet.y += bullet.velocityY;
                context.drawImage(bullet.image,
                        bullet.x - bullet.width/2, bullet.y - bullet.height/2);

                if(asteroid.isDead == false) {
                        var hit = intersects(
                           bullet.x - bullet.width/2, bullet.y - bullet.height/2,
                           bullet.width, bullet.height,
                           asteroid.x - asteroid.width/2, asteroid.y - asteroid.height/2,
                           asteroid.width, asteroid.height);
                        if(hit == true) {
                                bullet.isDead = true;
                                asteroid.isDead = true;
                        }
                }

                if(bullet.x < 0 || bullet.x > SCREEN_WIDTH ||
                        bullet.y < 0 || bullet.y > SCREEN_HEIGHT)
                {
                        bullet.isDead = true;
                }
        }

        // update the shoot timer
        if(shootTimer > 0)
                shootTimer -= deltaTime;

        // update all the bullets
        for(var i=0; i<bullets.length; i++)
        {
                bullets[i].x += bullets[i].velocityX;
                bullets[i].y += bullets[i].velocityY;
        }

        for(var i=0; i<bullets.length; i++)
        {
                // check if the bullet has gone out of the screen boundaries
                // and if so kill it
                if(bullets[i].x < -bullets[i].width ||
                        bullets[i].x > SCREEN_WIDTH ||
                        bullets[i].y < -bullets[i].height ||
                        bullets[i].y > SCREEN_HEIGHT)
                {
```

```
                        // remove 1 element at position i
                        bullets.splice(i, 1);
                        // because we are deleting elements from the middle of the
                        // array, we can only remove 1 at a time. So, as soon as we
                        // remove 1 bullet stop.
                        break;
                }
        }

        // draw all the bullets
        for(var i=0; i<bullets.length; i++)
        {
                context.drawImage(bullets[i].image,
                        bullets[i].x - bullets[i].width/2,
                        bullets[i].y - bullets[i].height/2);
        }

        . . .
}
```

Run this code and you should be able to shoot multiple bullets at once.

You may notice that even if you're holding down the fire button (space bar), as soon as you press another key you'll stop firing. See the exercises at the end of this tutorial for suggestions on how to fix this.

Re-adding Collisions:

Checking for collisions gets a bit tricker now. We don't just have one bullet and one asteroid – we have multiple bullets and multiple asteroids.

This means that each time through the run() function we need to check each bullet against each asteroid to see if there are any collisions.

Add the following code to the end of the run() function:

```
function run()
{
        . . .

        // check if any bullet intersects any asteroid. If so, kill them both
        for(var i=0; i<asteroids.length; i++)
        {
                for(var j=0; j<bullets.length; j++)
                {
                        if(intersects(
                                bullets[j].x - bullets[j].width/2, bullets[j].y –
                                        bullets[j].height/2,
                                bullets[j].width, bullets[j].height,
                                asteroids[i].x - asteroids[i].width/2, asteroids[i].y –
                                        asteroids[i].height/2,
                                asteroids[i].width, asteroids[i].height) == true)
                        {
                                asteroids.splice(i, 1);
                                bullets.splice(j, 1);
                                break;
                        }
                }
        }
}
```

Congratulations! You should now have a game that spawns multiple asteroids. Your ship will be able to fire multiple bullets at once, and if a bullet hits an asteroid they will both be destroyed.

Exercises:

You should be attempting the exercises each week, as they tutorials alone will not be enough for you to complete your game.

1. This week, make sure you are using delta time for all your movements. You will find that you will need to increase the speeds of all your objects as the speeds will now be in pixels per second.

2. Add the code from last week's exercise that makes the asteroid warp to the other side of the screen. There is a TODO: comment that indicates where you should add it.

3. Add the code from last week's exercises to kill the player if they collide with an asteroid.

4. You may have noticed a minor 'bug' in the shooting. Even if you have the space bar held, as soon as you press another key the ship stops firing. This is because the onKeyDown function is now being called with a different key press event.

To solve this, try making a variable (var shoot = false). Set this variable to true in the onKeyDown function, and to false in the onKeyUp function. Then, during the run() function, test the value of shoot and if it's true fire a new bullet.

5. Add a score to your game. When the player shoots an asteroid, increase the score. Use the text drawing functions to draw your score to the screen.