Tutorial – Finishing the Platformer

This week is all about getting done whatever needs to be done so that you have a working game.

If you're pressed for time, think about the minimum you need to have something that resembles a game (regardless of whether or not it's fun).

This tutorial is split up into sections, so you can pick and choose what you want/need to implement. You may already have got some this working on your own.

Enemies:

Enemies are actually easy. We'll just copy the code for the player, take out the bits for control via keyboard, and add some basic logic so that the enemies can move by themselves.

1. Add some enemy markers to a new layer in your level, wherever you want an enemy to appear.
   This is covered in the lecture slides for this lesson.
2. Copy the player.js file and rename it to enemy.js. Change the object name from Player to Enemy.
3. Add a reference to your enemy.js script in the HTML file
4. We can simplify the definition of an enemy to the following:

```
var Enemy = function(x, y)
{
	this.sprite = new Sprite("bat.png");
	this.sprite.buildAnimation(2, 1, 88, 94, 0.3, [0,1]);
	this.sprite.setAnimationOffset(0, -35, -40);

	this.position = new Vector2();
	this.position.set(x, y);

	this.velocity = new Vector2();

	this.moveRight = true;
	this.pause = 0;
}
```

   I'm just using 1 animation for my enemy, so I've also removed the constant variables relating to direction and the animation (I'll never change the animation since it only has one).
5. The Enemy update function is also very easy. It will keep moving in the current direction until it gets to either a wall or a cliff, then it will change direction (rinse, repeat).

© AIE 2015

If this kind of logic sounds good to you, and you don't think it7s something you could implement by yourself, then the following is the complete update code for the enemy:

```
Enemy.prototype.update = function(dt)
{
	this.sprite.update(dt);

	if(this.pause > 0)
	{
		this.pause -= dt;
	}
	else
	{
		var ddx = 0;                          // acceleration

		var tx = pixelToTile(this.position.x);
		var ty = pixelToTile(this.position.y);
		var nx = (this.position.x)%TILE;          // true if enemy overlaps right
		var ny = (this.position.y)%TILE;          // true if enemy overlaps below
		var cell = cellAtTileCoord(LAYER_PLATFORMS, tx, ty);
		var cellright = cellAtTileCoord(LAYER_PLATFORMS, tx + 1, ty);
		var celldown = cellAtTileCoord(LAYER_PLATFORMS, tx, ty + 1);
		var celldiag = cellAtTileCoord(LAYER_PLATFORMS, tx + 1, ty + 1);

		if(this.moveRight)
		{
			if(celldiag && !cellright) {
				ddx = ddx + ENEMY_ACCEL;      // enemy wants to go right

			}
			else {
				this.velocity.x = 0;
				this.moveRight = false;
				this.pause = 0.5;
			}
		}

		if(!this.moveRight)
		{
			if(celldown && !cell) {
				ddx = ddx - ENEMY_ACCEL;      // enemy wants to go left
			}
			else {
				this.velocity.x = 0;
				this.moveRight = true;
				this.pause = 0.5;
			}
		}

		this.position.x = Math.floor(this.position.x  + (dt * this.velocity.x));
		this.velocity.x = bound(this.velocity.x + (dt * ddx),
                                    -ENEMY_MAXDX, ENEMY_MAXDX);
	}
}
```

Everything from here out happens in main.js

© AIE 2015

6. In main.js add a few variables that were used in the enemy.js file. And of course create the enemies array.

   I've also added new layer constants. I haven't changed the value of LAYER_COUNT because I don't want to draw the enemy or trigger layers, and they aren't collision layers either.

```
var ENEMY_MAXDX = METER * 5;
var ENEMY_ACCEL = ENEMY_MAXDX * 2;

var enemies = [];

var LAYER_COUNT = 3;
var LAYER_BACKGOUND = 0;
var LAYER_PLATFORMS = 1;
var LAYER_LADDERS = 2;

var LAYER_OBJECT_ENEMIES = 3;
var LAYER_OBJECT_TRIGGERS = 4;
```

7. Update the initialize() function so that we read the enemy layer from the map and place an enemy on any tile that isn't empty.

   If you had more than one enemy type then you could place the correct type of enemy according to the value of the tile in the layer.

```
        // add enemies
        idx = 0;
        for(var y = 0; y < level1.layers[LAYER_OBJECT_ENEMIES].height; y++) {
                for(var x = 0; x < level1.layers[LAYER_OBJECT_ENEMIES].width; x++) {
                        if(level1.layers[LAYER_OBJECT_ENEMIES].data[idx] != 0) {
                                var px = tileToPixel(x);
                                var py = tileToPixel(y);
                                var e = new Enemy(px, py);
                                enemies.push(e);
                        }
                        idx++;
                }
        }
```

8. Now all that's left to do is to update and draw the enemies. I recommend doing these in separate steps, so update everything in your game and then draw everything.

   Here is a loop you can put in your run() function to update the enemies. Drawing the enemies is very similar (change the call to update to a call to draw)

© AIE 2015

```
        for(var i=0; i<enemies.length; i++)
        {
                enemies[i].update(deltaTime);
        }
```

Complete the code and you should now have enemies in your level.

Of course, if you want your player to die if they get hit by an enemy, you'll need to add that code yourself. The pseudo-code for this logic is included in the slides for this lesson. You would place the collision checking inside the update loop above.

```
for each enemy in enemies array
    update the enemy
    if player is alive
         if player's collision rectangle intersects the enemy's rect
             set player.isAlive to false
         end if
    end if
end for
```

Bullets:

Bullets are even easier. We'll just cover the main parts here, and the rest should be obvious.

Bullets will have a sprite (although not animated, you can use the sprite object to keep your drawing code consistent with the player and enemy), position, a velocity, and I also added a value to tell me if the bullet is moving left or right.

The complete code for the bullet is as follows:

```
var Bullet = function(x, y, moveRight)
{
        this.sprite = new Sprite("bullet.png");
        this.sprite.buildAnimation(1, 1, 32, 32, -1, [0]);
        this.sprite.setAnimationOffset(0, 0, 0);
        this.sprite.setLoop(0, false);

        this.position = new Vector2();
        this.position.set(x, y);

        this.velocity = new Vector2();

        this.moveRight = moveRight;
        if(this.moveRight == true)
                this.velocity.set(MAXDX *2, 0);
        else
                this.velocity.set(-MAXDX *2, 0);
}
```


© AIE 2015

```
Bullet.prototype.update = function(dt)
{
	this.sprite.update(dt);
	this.position.x = Math.floor(this.position.x  + (dt * this.velocity.x));
}

Bullet.prototype.draw = function()
{
	var screenX = this.position.x - worldOffsetX;
	this.sprite.draw(context, screenX, this.position.y);
}
```

You will need to create a bullets array in your main.js file, and use a loop to step through the array so you can update and draw every bullet (just like with the enemies).

Inside the update loop you should check each bullet for collisions against each enemy, and if the both collide you can destroy both the bullet and the enemy by removing them from their respective arrays.

You'll also need to kill the bullet if it ever goes off the screen.

Modify the run() function to implement these changes for updating the bullets:

```
	var hit=false;
	for(var i=0; i<bullets.length; i++)
	{
		bullets[i].update(deltaTime);
		if( bullets[i].position.x - worldOffsetX < 0 ||
			bullets[i].position.x - worldOffsetX > SCREEN_WIDTH)
		{
			hit = true;
		}

		for(var j=0; j<enemies.length; j++)
		{
			if(intersects( bullets[i].position.x, bullets[i].position.y, TILE, TILE,
				enemies[j].position.x, enemies[j].position.y, TILE, TILE) == true)
			{
				// kill both the bullet and the enemy
				enemies.splice(j, 1);
				hit = true;
				// increment the player score
				score += 1;
				break;
			}
		}
		if(hit == true)
		{
			bullets.splice(i, 1);
			break;
		}
	}
```


© AIE 2015

After you've added code to update and draw the bullets, the last thing to do is to shoot the bullets.

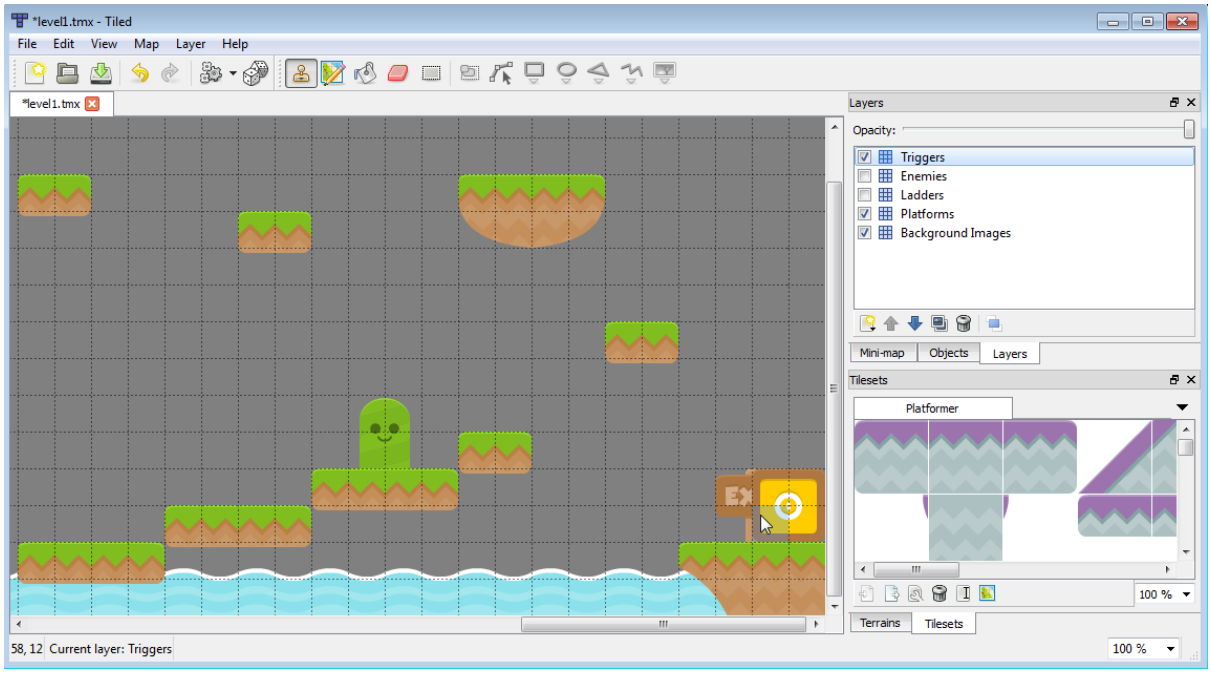
You've already updated the player's update function to play a sound effect when pressing the space bar (in last week's tutorial). All that needs to be done is, at the same location in code, create a bullet, set it to the same location as the player, set which direction it's moving in, and then add it to the end of the bullets array.

Hopefully by now you have enough understanding of your code and how to program to make these changes yourself. If you get lost, ask your teacher or send an email to helponline@aie.edu.au.

Triggers:

The last thing that will be covered in this tutorial is triggers. Adding a trigger at the end of the level will allow you to switch to the Game Over state. You may also want to put other triggers in your level (maybe your player will die if they walk on a spike trap, for example).

Create a new layer in your map, just like we did for the enemies. Put a tile at the end of your level to use as the game-over trigger. When the player collides with this tile then we'll switch to the game-over state.



Export your map and update the level1.js file.

In the initialize() function in main.js, we want to read the new Trigger layer and update our collision map. That is, we want a new entry in our collision map for this new layer.

The code to add to your initialize function looks like this:

© AIE 2015

```
// initialize trigger layer in collision map
cells[LAYER_OBJECT_TRIGGERS] = [];
idx = 0;
for(var y = 0; y < level1.layers[LAYER_OBJECT_TRIGGERS].height; y++) {
	cells[LAYER_OBJECT_TRIGGERS][y] = [];
	for(var x = 0; x < level1.layers[LAYER_OBJECT_TRIGGERS].width; x++) {
		if(level1.layers[LAYER_OBJECT_TRIGGERS].data[idx] != 0) {
			cells[LAYER_OBJECT_TRIGGERS][y][x] = 1;
			cells[LAYER_OBJECT_TRIGGERS][y-1][x] = 1;
			cells[LAYER_OBJECT_TRIGGERS][y-1][x+1] = 1;
			cells[LAYER_OBJECT_TRIGGERS][y][x+1] = 1;
		}
		else if(cells[LAYER_OBJECT_TRIGGERS][y][x] != 1) {
			// if we haven't set this cell's value, then set it to 0 now
			cells[LAYER_OBJECT_TRIGGERS][y][x] = 0;
		}
		idx++;
	}
}
```

You'll probably see similarities between this code and that for our regular collision layers. That's because most of it is the same.

To tell when the player has entered the trigger, you can add the following line of code to the end of the player's update function:

```
if(cellAtTileCoord(LAYER_OBJECT_TRIGGERS, tx, ty) == true)
{
	// game over man, game over
}
```

If you've implemented your game states already, then this is when you'd switch to the game-over state.

We have now covered everything you'll need to include for a basic, passable, platformer game. This lesson's tutorial has intentionally been a bit more vague, because by now you should have the knowledge to fill in the blanks.

If you need further clarification on anything presented in this tutorial, or anything covered in the course so far, don't hesitate to let your teacher know.

Exercises:

Implement the ladder climbing as described in the slides for this lesson. This will involve adding states to the player's logic (a switch statement in the player's update function would suffice here).

© AIE 2015