**Creating a game like Tanks / Worms, Part 1**

*By*[*Richard Davey*](https://twitter.com/photonstorm) on 9th January 2015   [@photonstorm](https://twitter.com/photonstorm)

**Welcome!**

If you're anything like me you probably write *loads* of code. From prototypes demonstrating a single mechanic to helper functions. And sometimes you might even finish a game :)

I started this series as a means to share code with you in an informal manner, one game dev to another. The aim is to build-up something over the course of a couple of weeks. Each chunk being easily digestible in a short period of time. It's something I call "lunch break game dev": in that you can read and experiment for yourself in under an hour (just try not to get crumbs in your keyboard.)

**IT'S SOMETHING I CALL "LUNCH BREAK GAME DEV": IN THAT YOU CAN READ AND EXPERIMENT FOR YOURSELF IN UNDER AN HOUR**

I'm not writing to any fixed agenda here. I fully expect this series to evolve. If something doesn't work I'll pull it, and I equally won't be scared of trying new things. Maybe one week the code will be far too easy for you, another it might be the opposite. But hopefully in all cases it will spark some ideas for you to play with regardless.

Finally I'm not here to try and "sell" you anything. If I release a new book or plugin then I'll probably write about it, but only because I'll be so pleased at having achieved that! On the whole this is our place to write some code together, so please do leave feedback and ideas.

**Tanks - Part 1**

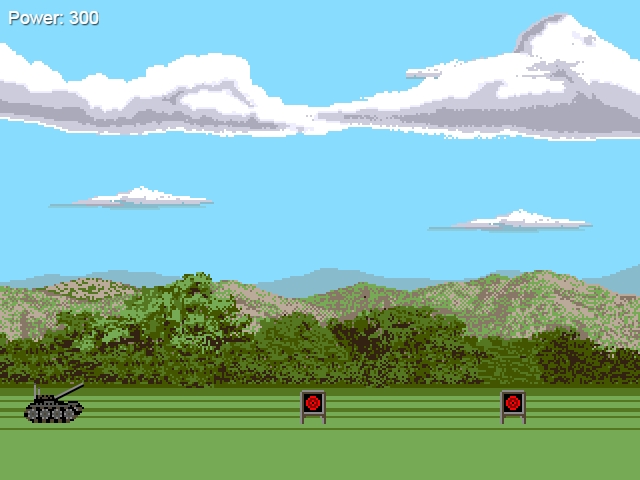
Let's make a [Tanx](https://www.youtube.com/watch?v=XuBWZEaZAI0" \o "Amiga Tanx) style game. It uses the same mechanic as games like [QBASIC Gorillas](https://www.youtube.com/watch?v=UDc3ZEKl-Wc) and Scorched Earth.

You control a tank. You can adjust the elevation of the turret, set the power and then fire. The bullet flies across the landscape hopefully hitting the targets.

This week we'll display the tank, allow it to aim, fire and track the bullet with the camera. Next week we'll add the landscape and allow you to blow holes in it!

**Get the source**

Run and live-edit the code on [jsbin](http://jsbin.com/fasaku/3/edit?js,output) or [codepen](http://codepen.io/photonstorm/pen/gbgvpM?editors=001). You can also clone the [git repo](https://github.com/photonstorm/phaser-coding-tips).



*Graphics from Amiga Tanx Copyright 1991 Gary Roberts*

**init**

Because this game uses pixel art we're going to use a rounded canvas renderer:

this.game.renderer.renderSession.roundPixels = true;

This will stop Phaser from rendering graphics at sub-pixel locations, keeping them nice and crisp. We'll also set the game world to be 992 pixels wide, enable physics and setting a gravity value of 200:

this.game.world.setBounds(0, 0, 992, 480);

this.physics.startSystem(Phaser.Physics.ARCADE);

this.physics.arcade.gravity.y = 200;

**create**

We need to assemble the tank. It's split into two images: the base of the tank, and the turret. The turret is positioned against the base so it looks correct when rotating. The area highlighted in red is where they are "joined":



We also create a Group of targets to shoot. This is a standard Phaser Group with Arcade Physics enabled on it. However because of this we need to stop gravity from pulling the targets away:

this.targets.setAll('body.allowGravity', false);

setAll lets you quickly set the same property across all members of the Group. In this case we tell it to disable gravity.

The rest of the create block is simply putting together sprites, text and some basic keyboard controls.

**Set the turret angle**

In the update method we allow the player to set the angle of the turret. This is done with a check to ensure it is kept within limits and then we change the Sprite.angle property of the turret. The default anchor of the turret means this rotation works correctly with no further settings.

if (this.cursors.up.isDown && this.turret.angle > -90)

{

this.turret.angle--;

}

else if (this.cursors.down.isDown && this.turret.angle < 0)

{

this.turret.angle++;

}

**fire**

When the player presses the space bar we'll fire a shot. This is handled in the fire method. It starts by setting the bullet back to the turret coordinates (in case it has already been fired):

this.bullet.reset(this.turret.x, this.turret.y);

We also want to display the 'flame' sprite when they shoot. This is a burst of fire that emits from the end of the turrets gun then fades away:



We know the coordinates of the left of the turret, but what about the end? There are several ways to solve this but I chose to use Point.rotate.

var p = new Phaser.Point(this.turret.x, this.turret.y);

p.rotate(p.x, p.y, this.turret.rotation, false, 34);

this.flame.x = p.x;

this.flame.y = p.y;

This allows you to calculate where the Point would be if it was rotated and moved from its origin. In the code above we set the rotation to match the turret, and the distance 34 pixels works for these assets. The end result is that the flame effect appears at the end of the gun, regardless of its angle of rotation.

The final part of firing is to launch the bullet.

this.camera.follow(this.bullet);

this.physics.arcade.velocityFromRotation(this.turret.rotation, this.power, this.bullet.body.velocity);

The launch trajectory is based on the angle of the turret and the power the player has set. It will calculate the velocity need for these two factors and inject them into the velocity of the bullet.

At the same time we tell the Camera to track the bullet as it flies.

**boom?**

In the update method we check if the bullet exists (i.e. is in flight), and if so we perform an overlap check between it and the targets. If they overlap the target is killed and the bullet removed:

removeBullet: function () {

this.bullet.kill();

this.camera.follow();

this.add.tween(this.camera).to( { x: 0 }, 1000, "Quint", true, 1000);

}

We need to stop the Camera tracking the bullet so that the tween works. The tween pauses for 1 second then tweens the Camera back to look at the tank again ready for the next shot. If you don't stop the Camera following the tween will seem to fail, because Camera tracking takes priority over positioning of it.

That's it for Part 1. Obviously there's more code in the source, but all the important bits are covered above.

**In Part 2**

In Part 2 we'll add a complete landscape. The targets will be positioned all over it. And then we'll add the code needed to let you blow holes out of it all and have the terrain deform in real-time.

## Creating a game like Tanks / Worms, Part 2

By[*Richard Davey*](https://twitter.com/photonstorm) on 16th January 2015   [@photonstorm](https://twitter.com/photonstorm)

### Welcome!

Missed [Issue 1](http://phaser.io/tutorials/coding-tips-001/)? Read it before continuing. It contains Part 1 of the Tanks game.

I was really pleased with the reception the first issue received! It seems I wasn't the only one looking for small and focused tutorials. In this issue we complete the tanks game, although there's plenty of scope left for you to push it further. If you do so please share the end result!

### Tanks - Part 2

In Part 1 we got the tank displayed and firing a single bullet at some targets. Now it's time to make it a proper tanks game and add a landscape!



Graphics from Amiga Tanx Copyright 1991 Gary Roberts

### Get the source

I'd suggest you have a play of the game and a quick scroll through the code. I'm only going to highlight the most important parts here, or those that need more explanation. If you've questions about a part I didn't include then please use the forum to ask it.

Run and live-edit the code on [jsbin](http://jsbin.com/badime/1/edit?js,output) or [codepen](http://codepen.io/photonstorm/pen/ogZJPP?editors=001). You can also clone the [git repo](https://github.com/photonstorm/phaser-coding-tips).

### create

The create method is almost identical to Part 1. The difference is that the targets are now positioned so they lay on the new landscape.

We've also added two new Game Objects: the land and a particle emitter. The land is a BitmapData object to which we draw our land.png file. This PNG has the landscape drawn on a transparent background. It's drawn exactly as it appears in the screen shot above (feel free to edit it!)

this.land = this.add.bitmapData(992, 480);

this.land.draw('land');

this.land.update();

this.land.addToWorld();

After we've drawn the PNG to the BitmapData we have to update it. This is because we need to access its pixel data during the game. The final line adds it to the game world. Internally this creates a new Sprite object, sets the BitmapData to be its texture and adds it to the Game World at 0, 0 (because we didn't specify any other location).

That's all we need to visually add the land, but how do we destroy it?

### bulletVsLand()

If you look at the update method you'll see that we're checking if the bullet exists. If so we check it against the targets and then the land:

if (this.bullet.exists)

{

this.physics.arcade.overlap(this.bullet, this.targets, this.hitTarget, null, this);

this.bulletVsLand();

}

The bulletVsLand method starts with a simple bounds check. If the bullet goes out of bounds then we kill it and return (as it can't now hit the land). Notice that we don't check the 'top' of the world, as we want it to be allowed to rise up above the screen and fall back down again.

if (this.bullet.x < 0 || this.bullet.x > this.game.world.width || this.bullet.y > this.game.height)

{

this.removeBullet();

return;

}

The next part is the meaty bit:

var x = Math.floor(this.bullet.x);

var y = Math.floor(this.bullet.y);

var rgba = this.land.getPixel(x, y);

if (rgba.a > 0)

{

this.land.blendDestinationOut();

this.land.circle(x, y, 16, 'rgba(0, 0, 0, 255');

this.land.blendReset();

this.land.update();

this.removeBullet();

}

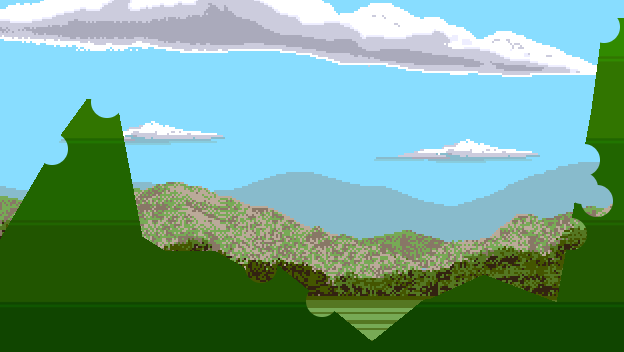
Because we're going to be doing a pixel color look-up on the BitmapData we have to floor the bullet coordinates. Once done we can use the BitmapData.getPixel method to get a Color object for the given pixel. This is done every frame as the bullet flies through the air, we sample the pixel color beneath it.

Our PNG is a landscape drawn on a transparent background, so all we need to do is check that we're over a pixel that has an alpha value greater than zero. If this is the case we blow a chunk out of the land.

This is done by setting the destination-out blend mode. If you draw on a canvas with this blend mode you can effectively "remove" parts of it. In this case we'e drawing a 16px sized circle where the bullet landed. Combine this with the blend mode and you punch a small hole into the land.

The final few lines reset the blend mode and call BitmapData.update which tells it to rescan the pixel data and render the new scene. Finally the bullet is removed, its job done.

Using this approach you can soon make Swiss-cheese of the landscape:



Feel free to vary the circle size! Or even draw an entirely different shape.

### Great balls of fire

With the land reacting to our bullets we can ice this cake by adding an explosion effect when we hit a target. For this we'll use an Emitter. We make it in the create method:

this.emitter = this.add.emitter(0, 0, 30);

this.emitter.makeParticles('flame');

this.emitter.setXSpeed(-120, 120);

this.emitter.setYSpeed(-100, -200);

this.emitter.setRotation();

It's re-using the flame.png which we use for the tank fire effect. Rotation is disabled by calling setRotation with no parameters. When the particles emit they'll pick a random x velocity between -120 and 120, and a vertical one between -100 and -200 (thrusting up into the air).

To activate the emitter we call it in the hitTarget method:

hitTarget: function (bullet, target) {

this.emitter.at(target);

this.emitter.explode(2000, 10);

target.kill();

this.removeBullet(true);

}

The emitter is positioned on the center of the target Sprite and set to explode. The first parameter tells the flames to live for 2 seconds, the 2nd to explode 10 of them at once. Then we kill the target and the bullet.

This creates a suitably explosive effect:



You may notice that this call to removeBullet passes a value of true. This tells the Camera tween to delay for a little longer before returning to the tank. It gives you a little more time to enjoy the effect :)

### Ideas for improvements

Well, where do I start?! There are hundreds of things you could do to this game. Taking inspiration from both the original and genre evolutions like Worms you could add all manner of fun:

* An AI opponent to shoot back at you!
* Add a random wind effect, throwing your bullet off course
* New weapons :) (Holy hand grenades anyone?)
* Randomly create the landscape instead of using a PNG

Whatever you decide to do hopefully this has given you a taster and some brain food. If you evolve this further I'd love to know!