Advanced FPS Kit
Tutorials
Weapon Design & Implementation

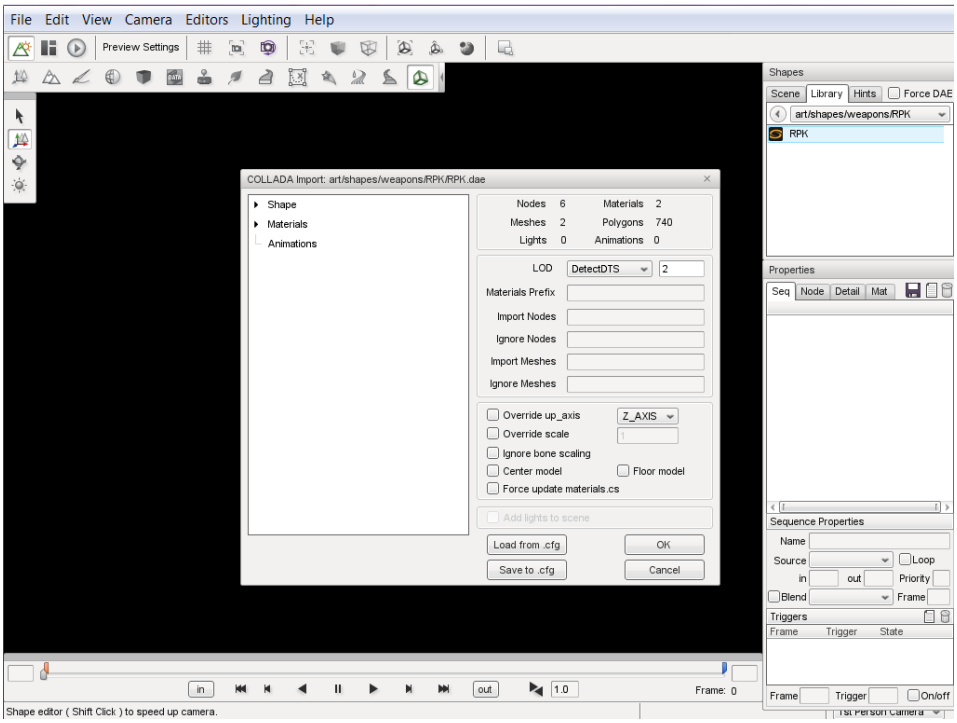
Welcome to my tutorial on Weapon Design & Implementation. For those of you who have been following along on the Weapon Modelling tutorial, you can consider this the "Unofficial" Part 5 of the series, as I will show you how to bring a gun into the engine for use in the engine. This tutorial will also go over some general guidelines on weapon design for your games to help you out and to show you how to accomplish some very interesting things for your weapons.

Part 1: Loading a Completed Model into Torque 3D

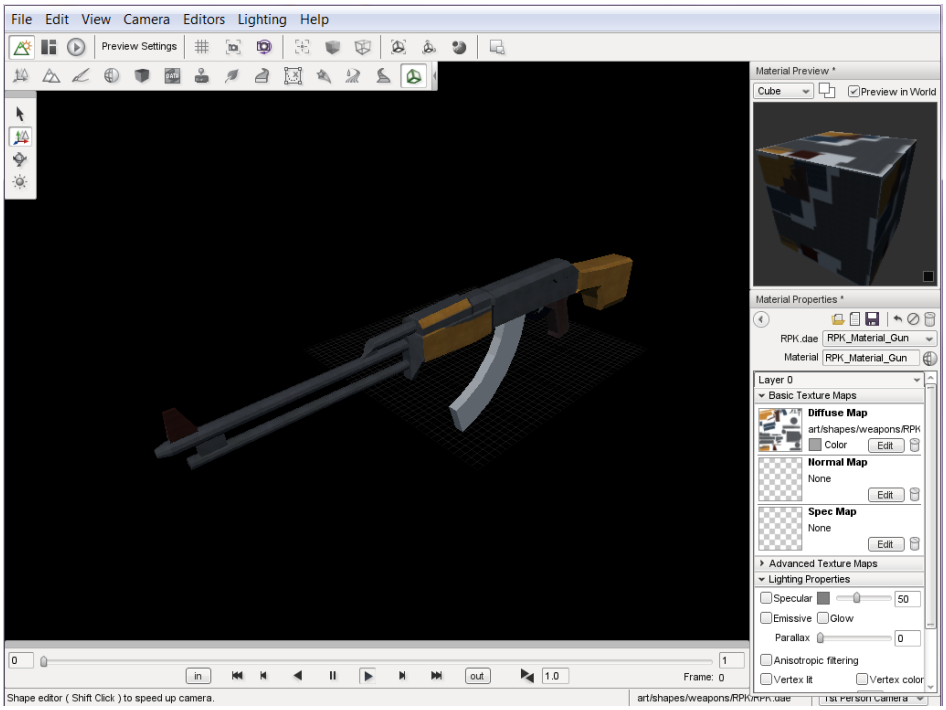
Without further introductions or conversations, let's jump right into the action. So, for those of you who didn't follow along, we just got done building a RPK Machine Gun model last time around and it turned out like so:

While this isn't your ultra-detailed super-model gun, this still is a "gun" and it will still pack quite a punch if coded properly. But how exactly do I get my gun into the engine? Well, it's not all that difficult. Go ahead and get your .DAE file (If you haven't exported the model yet, do so now) and place it in the shapes folder of your game's art folder. For example, I have: art/shapes/weapons/RPK/files

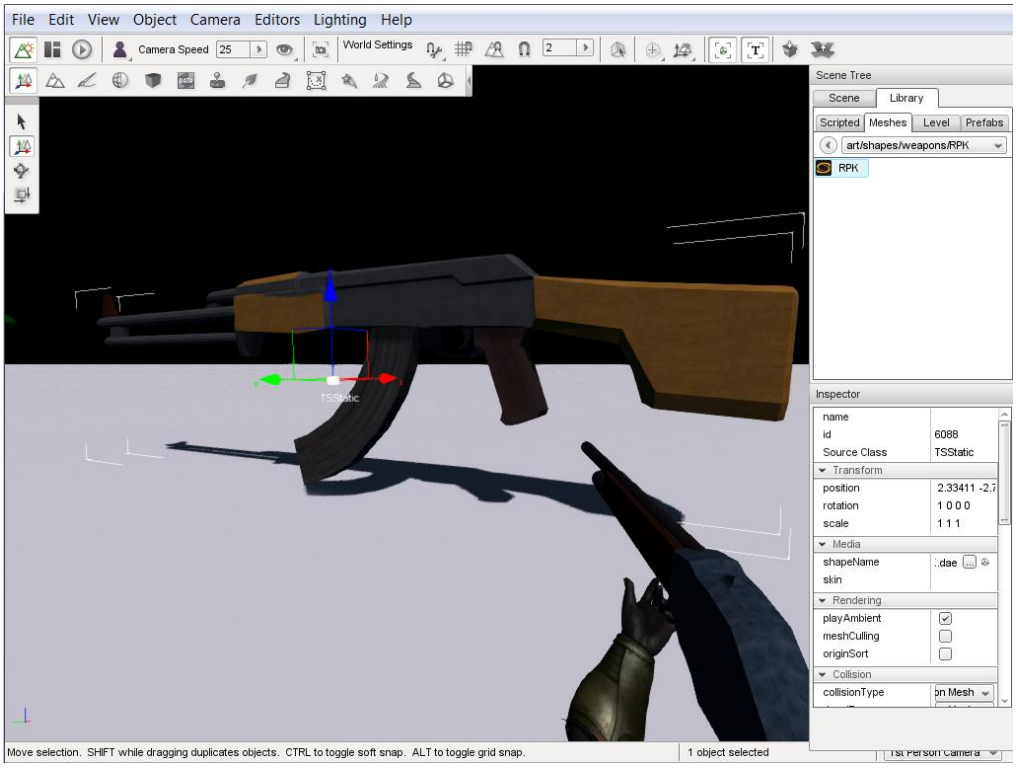
Once in there, load up your engine and hop into the world editor. You'll select the Shape Editor, then in the shape editor under the library tab, navigate to your .DAE file and load it in using the default options:



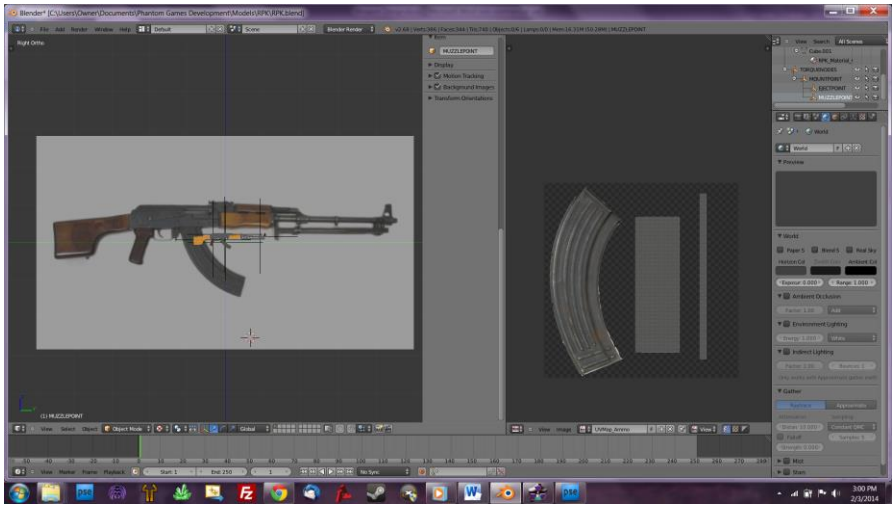
The model will now load in without any textures on it. Now, under the properties section select the Mat (Materials) tab. You should see all of the materials associated with that weapon listed there. Select the material you want to associate and select the edit button. From there you can add diffuse, specular, and normal maps to the material. Once you've added all you need, save the material.



The next thing we need to do is ensure the weapon is “fit-to-scale”. So go back to the world editor and try to place an instance of the weapon model in the world.



In the case of the modelling tutorial, and about 90% of the time, your model will be “way” out of size. No problem though, with the COLLADA pipeline, you can re-open blender and scale everything down in edit mode, then re-export without needing to do anything else, and the best part is your changes can be seen in real time. So scale it down appropriately. Another good tip is to load in a model you’ve already scaled alongside the current model, scale it down, then delete the reference. For example, here’s the true size, versus the reference size:



You can see how the reference image is considerably larger than the actual model itself. So, go ahead and fix the model if necessary, then bring it back into the engine:

Now that's a lot better. From here, the engine will generate for you a .cached.dts file, when you ship your game, this is the file you will send out. Don't send out the .DAE, unless you want people to be able to use your art assets anywhere, for weapon models you get from paid packs, you will be required to delete the DAE file if they send you one. As for now though, you can close Torque 3D since your work in the engine is finished for now.

NOTE: If you reloaded the model file, it will load in the DAE, therefore do not delete the DAE immediately, you will need to generate a new .dts file by loading any mission once after you make changes to the DAE.

Part 2: Scripting Weapons

Now comes the fun part, actually "making" the weapon, act like a weapon. Now, there are many types of different weapons out there, machine guns, assault rifles, sniper rifles, explosives, ect. But it can only do what you tell it to do, and this is where you will need to flex your TorqueScripting muscles. Now, I've included in this tutorial, my Model 1887 Shotgun model for your use (If you don't own GarageGames' Soldier Art Pack, you'll need to incorporate you own sound). I've also included my TorqueScript file for you to learn from. So feel free to mess around with that as you please. This tutorial will be about the RPK.

So, let's go ahead and make a new script file on the server-side. I personally, am against with a passion, the separation of datablocks and script files, my 9 years of Torque Experience with Tribes 2 has taught me that datablocks and scripts go together

like bread and butter, they are inseparable to me. First off, let's decide how we want the RPK to behave. The RPK is a light machine gun, so it needs to be able to put down a lot of rounds before having to reload. I'll say a good clip size is about 50 for this. Next up is the damage factor of the gun. For balancing purposes (see my segment later on, or read my FPS Design Tutorial), I want to keep the RPK as an excellent weapon of choice for medium-long range suppression. So, to do this, we'll give it fairly moderate damage with moderate recoil. I'm thinking somewhere around 22 Damage per Bullet will work nicely. So, let's start our script file with what we have:

```
//--------------------------------------------------------------------------
// RPK.cs
// I <3 Machine Guns
//--------------------------------------------------------------------------

datablock SFXProfile(RPKFireSound) {
   filename = "art/sound/turret/wpn_turret_fire";
   description = AudioClose3D;
   preload = true;
};

// ----------------------------------------------------------------------------
// Particles
// ----------------------------------------------------------------------------

//--------------------------------------------------------------------------
// Explosion
//--------------------------------------------------------------------------

//--------------------------------------------------------------------------
// Projectile Object
//--------------------------------------------------------------------------
datablock ProjectileData( RPKProjectile ) {
   projectileShapeName = "";

   directDamage        = 22;
   radiusDamage        = 0;
   damageRadius        = 0.5;
   areaImpulse         = 0.5;
   impactForce         = 1;

   explosion           = BulletDirtExplosion;
   decal               = BulletHoleDecal;

   muzzleVelocity      = 120;
   velInheritFactor    = 1;

   armingDelay         = 0;
   lifetime            = 350;
   fadeDelay           = 972;
   bounceElasticity    = 0;
   bounceFriction      = 0;
   isBallistic         = false;
   gravityMod          = 1;
```

```
};

function RPKProjectile::onCollision(%this,%obj,%col,%fade,%pos,%normal) {
   // Apply impact force from the projectile.
   // Apply damage to the object all shape base objects
   if ( %col.getType() & $TypeMasks::GameBaseObjectType ) {
      %col.damage(%obj, %pos, %this.directDamage, "RPK");
   }
}
```

Feel free to tweak these values to whatever you deem necessary for your weapon, remember, this is your gun, so you will code it the way you want it to be coded. Now, let's add out clip and ammo box for the gun

```
//--------------------------------------------------------------------------
// Ammo Item
//--------------------------------------------------------------------------
datablock ItemData(RPKClip) {
   // Mission editor category
   category = "AmmoClip";
   className = "AmmoClip";

   // Basic Item properties
   shapeFile = "art/shapes/weapons/RPK/RPK.DAE";
   mass = 1;
   elasticity = 0.2;
   friction = 0.6;

   // Dynamic properties defined by the scripts
   pickUpName = "RPK clip";
   count = 1;
   maxInventory = 10;
};

datablock ItemData(RPKAmmo) {
   // Mission editor category
   category = "Ammo";
   className = "Ammo";

   // Basic Item properties
   shapeFile = "art/shapes/weapons/RPK/RPK.DAE";
   mass = 1;
   elasticity = 0.2;
   friction = 0.6;

   // Dynamic properties defined by the scripts
   pickUpName = "RPK ammo";
   maxInventory = 50;
   clip = RPKClip;
};
```

Notice how I left the file path as .DAE in the end? Well, Torque 3D knows to check for the .dts file if the DAE isn't found, so we'll leave that as so. Next up we'll code the gun

itself. Also a quick note to those who followed along with the Model tutorial, you can just as easily delete the RPK “Gun” object, and export just the clip of the gun to create a separate model for the Ammo and Clip objects above, the choice is yours.

So now we’ll code the Weapon item and the Weapon Image:

```
//--------------------------------------------------------------------------
// Weapon Item
//--------------------------------------------------------------------------
datablock ItemData(RPK) {
   // Mission editor category
   category = "Weapon";
   className = "Weapon";

   // Basic Item properties
   shapeFile = "art/shapes/weapons/RPK/RPK.DAE";
   mass = 1;
   elasticity = 0.2;
   friction = 0.6;
   emap = true;

   // Dynamic properties defined by the scripts
   PreviewImage = "RPK.png";
   pickUpName = "RPK";
   description = "RPK";
   image = Model1887WeaponImage;
   reticle = "crossHair";
   heldWeaponName = "RPK Machine Gun";
};


datablock ShapeBaseImageData(RPKWeaponImage) {
   // Basic Item properties
   shapeFile = "art/shapes/weapons/RPK/RPK.DAE";
   shapeFileFP = "art/shapes/weapons/RPK/RPK.DAE";
   emap = true;

   imageAnimPrefix = "Rifle";
   imageAnimPrefixFP = "Rifle";

   mountPoint = 0;
   firstPerson = true;
   useEyeNode = true;
   animateOnServer = true;

   correctMuzzleVector = true;

   class = "WeaponImage";
   className = "WeaponImage";

   ammoImage = "LMG";

   // Projectiles and Ammo.
```

```
item = RPK;
ammo = RPKAmmo;
clip = RPKClip;

projectile = RPKProjectile;
projectileType = Projectile;
projectileSpread = "0.05";
projectileNum = 1;

casing = BulletShell;
shellExitDir        = "1.0 0.3 1.0";
shellExitOffset     = "0.15 -0.56 -0.1";
shellExitVariance   = 15.0;
shellVelocity       = 3.0;

// Weapon lights up while firing
lightType = "WeaponFireLight";
lightColor = "0.992126 0.968504 0.708661 1";
lightRadius = "8";
lightDuration = "100";
lightBrightness = 2;

// Shake camera while firing.
shakeCamera = true;
camShakeFreq = "0 0 0";
camShakeAmp = "0 0 0";

useRemainderDT = true;

// Initial start up state
stateName[0]                    = "Preactivate";
stateTransitionOnLoaded[0]      = "Activate";
stateTransitionOnNoAmmo[0]      = "NoAmmo";

// Activating the gun.  Called when the weapon is first
// mounted and there is ammo.
stateName[1]                    = "Activate";
stateTransitionGeneric0In[1]    = "SprintEnter";
stateTransitionOnTimeout[1]     = "Ready";
stateTimeoutValue[1]            = 0.5;
stateSequence[1]                = "switch_in";
stateSound[1]                   = LurkerSwitchinSound;

// Ready to fire, just waiting for the trigger
stateName[2]                    = "Ready";
stateTransitionGeneric0In[2]    = "SprintEnter";
stateTransitionOnMotion[2]      = "ReadyMotion";
stateTransitionOnTimeout[2]     = "ReadyFidget";
stateTimeoutValue[2]            = 10;
stateWaitForTimeout[2]          = false;
stateScaleAnimation[2]          = false;
stateScaleAnimationFP[2]        = false;
stateTransitionOnNoAmmo[2]      = "NoAmmo";
stateTransitionOnTriggerDown[2]  = "Fire";
```

```
stateSequence[2]                = "idle";

// Same as Ready state but plays a fidget sequence
stateName[3]                    = "ReadyFidget";
stateTransitionGeneric0In[3]    = "SprintEnter";
stateTransitionOnMotion[3]      = "ReadyMotion";
stateTransitionOnTimeout[3]     = "Ready";
stateTimeoutValue[3]            = 6;
stateWaitForTimeout[3]          = false;
stateTransitionOnNoAmmo[3]      = "NoAmmo";
stateTransitionOnTriggerDown[3] = "Fire";
stateSequence[3]                = "idle_fidget1";
stateSound[3]                   = LurkerIdleSound;

// Ready to fire with player moving
stateName[4]                    = "ReadyMotion";
stateTransitionGeneric0In[4]    = "SprintEnter";
stateTransitionOnNoMotion[4]    = "Ready";
stateWaitForTimeout[4]          = false;
stateScaleAnimation[4]          = false;
stateScaleAnimationFP[4]        = false;
stateSequenceTransitionIn[4]    = true;
stateSequenceTransitionOut[4]   = true;
stateTransitionOnNoAmmo[4]      = "NoAmmo";
stateTransitionOnTriggerDown[4] = "Fire";
stateSequence[4]                = "run";

// Fire the weapon. Calls the fire script which does
// the actual work.
stateName[5]                    = "Fire";
stateTransitionGeneric0In[5]    = "SprintEnter";
stateTransitionOnTimeout[5]     = "NewRound";
stateTimeoutValue[5]            = 0.15;
stateFire[5]                    = true;
stateRecoil[5]                  = LightRecoil;
stateAllowImageChange[5]        = false;
stateSequence[5]                = "fire";
stateScaleAnimation[5]          = false;
stateSequenceNeverTransition[5] = true;
stateSequenceRandomFlash[5]     = true;        // use muzzle flash sequence
stateScript[5]                  = "onFire";
stateSound[5]                   = RPKFireSound;
stateEmitter[5]                 = GunFireSmokeEmitter;
stateEmitterTime[5]             = 0.025;

// Put another round into the chamber if one is available
stateName[6]                    = "NewRound";
stateTransitionGeneric0In[6]    = "SprintEnter";
stateTransitionOnNoAmmo[6]      = "NoAmmo";
stateTransitionOnTimeout[6]     = "Ready";
stateWaitForTimeout[6]          = "0";
stateTimeoutValue[6]            = 0.15;
stateAllowImageChange[6]        = false;
stateEjectShell[6]              = true;
```

```
   // No ammo in the weapon, just idle until something
   // shows up. Play the dry fire sound if the trigger is
   // pulled.
   stateName[7]                    = "NoAmmo";
   stateTransitionGeneric0In[7]     = "SprintEnter";
   stateTransitionOnMotion[7]       = "NoAmmoMotion";
   stateTransitionOnAmmo[7]         = "ReloadClip";
   stateTimeoutValue[7]             = 0.1;   // Slight pause to allow script to run when trigger is still held down
from Fire state
   stateScript[7]                  = "onClipEmpty";
   stateSequence[7]                 = "idle";
   stateScaleAnimation[7]           = false;
   stateScaleAnimationFP[7]         = false;
   stateTransitionOnTriggerDown[7]  = "DryFire";

   stateName[8]                    = "NoAmmoMotion";
   stateTransitionGeneric0In[8]     = "SprintEnter";
   stateTransitionOnNoMotion[8]     = "NoAmmo";
   stateWaitForTimeout[8]           = false;
   stateScaleAnimation[8]           = false;
   stateScaleAnimationFP[8]         = false;
   stateSequenceTransitionIn[8]     = true;
   stateSequenceTransitionOut[8]    = true;
   stateTransitionOnTriggerDown[8]  = "DryFire";
   stateTransitionOnAmmo[8]         = "ReloadClip";
   stateSequence[8]                 = "run";

   // No ammo dry fire
   stateName[9]                    = "DryFire";
   stateTransitionGeneric0In[9]     = "SprintEnter";
   stateTransitionOnAmmo[9]         = "ReloadClip";
   stateWaitForTimeout[9]           = "0";
   stateTimeoutValue[9]             = 0.7;
   stateTransitionOnTimeout[9]      = "NoAmmo";
   stateScript[9]                  = "onDryFire";
   stateSound[9]                    = MachineGunDryFire;

   // Play the reload clip animation
   stateName[10]                    = "ReloadClip";
   stateTransitionGeneric0In[10]     = "SprintEnter";
   stateTransitionOnTimeout[10]      = "Ready";
   stateWaitForTimeout[10]           = true;
   stateTimeoutValue[10]             = 3.0;
   stateReload[10]                  = true;
   stateSequence[10]                 = "reload";
   stateShapeSequence[10]            = "Reload";
   stateScaleShapeSequence[10]       = true;
   stateSound[10]                    = LurkerReloadSound;

   // Start Sprinting
   stateName[11]                    = "SprintEnter";
   stateTransitionGeneric0Out[11]   = "SprintExit";
   stateTransitionOnTimeout[11]     = "Sprinting";
```

```
   stateWaitForTimeout[11]         = false;
   stateTimeoutValue[11]           = 0.5;
   stateWaitForTimeout[11]         = false;
   stateScaleAnimation[11]         = false;
   stateScaleAnimationFP[11]       = false;
   stateSequenceTransitionIn[11]   = true;
   stateSequenceTransitionOut[11]  = true;
   stateAllowImageChange[11]       = false;
   stateSequence[11]               = "sprint";

   // Sprinting
   stateName[12]                   = "Sprinting";
   stateTransitionGeneric0Out[12]  = "SprintExit";
   stateWaitForTimeout[12]         = false;
   stateScaleAnimation[12]         = false;
   stateScaleAnimationFP[12]       = false;
   stateSequenceTransitionIn[12]   = true;
   stateSequenceTransitionOut[12]  = true;
   stateAllowImageChange[12]       = false;
   stateSequence[12]               = "sprint";

   // Stop Sprinting
   stateName[13]                   = "SprintExit";
   stateTransitionGeneric0In[13]   = "SprintEnter";
   stateTransitionOnTimeout[13]    = "Ready";
   stateWaitForTimeout[13]         = false;
   stateTimeoutValue[13]           = 0.5;
   stateSequenceTransitionIn[13]   = true;
   stateSequenceTransitionOut[13]  = true;
   stateAllowImageChange[13]       = false;
   stateSequence[13]               = "sprint";
};
```

Now the Weapon Image is where you'll define all of the characteristics of your weapon, for example a shotgun will have a state for firing, re-cocking the shotgun (for pump action), reloading single bullets, ect. Whereas a machine gun will behave similarly to the Lurker Assault Rifle, with a longer reload time. All of the weapon properties defined here will be used by the gun when individual actions are handled by the gun. Go ahead and save your script file, and be sure to execute it on the server. If you get complaints about missing datablocks, then code it up to meet the style of the horrible T3D scheme of datablock/script seperat— I'm just not going there. Either way, we'll move onto the next part.

The next step here is to add the gun to the Player's datablock max inventory array so the player is allowed to carry the gun. You will see definitions near the bottom of the player datablock for MaxInv relating to your guns. Just add an entry for the RPK like so:

```
   maxInv[RPK] = 1;
   maxInv[RPKClip] = 10;
```

Now if you own the full AFPSK with the custom loadout tutorial, feel free to follow the instructions there to add the RPK to your loadout list, otherwise you'll have to either resort to the T3D weapon cycle system, or just to give the RPK to the player with playerID.addInventory(RPK, 1); and playerID.addInventory(RPKClip, 10); to give your player the RPK and the associated clip. At this point you're ready to fire up your game and use your shiny new machine gun, go have fun!

Part 3: Weapon Design & Development

This next part of the tutorial is going to be all about weapon design techniques, balancing and talking about development of more complex weapon designs.

So, as stated in the FPS Design Tutorial, Weapon Balance is one of the most important factors of a good FPS game. You want to ensure that each player's weapons augment the player's style of gaming, while catering to the fact that you don't want to create weapon setups that would be "overpowered" compared to others.

I'll give you an example of a game gone horribly wrong here, and I'm sure you've played it at least once. Call of Duty. This game, has more weapon design flaws than any other shooter game I've seen or played for that manner. We'll take a very good example of their flawed design logic. A Shotgun player for example, enjoys to run around the map at speeds as fast as possible in order to get as close as possible to land the one hit kill at a close distance, which is what a shotgun should do if they're close enough. Now, we'll go to the other spectrum. A Sniper player "should" be sitting back, trying to avoid

the combat altogether and instead of being the run and gun player, playing more of a support role for the team. Unfortunately, Call of Duty has introduced a game flaw in which snipers have an advantage at both ranges. If the game were truly balanced with this design flaw, shotguns should as well have the range effect and land one hit kills at long range, but they don't, not in real life, and not in Call of Duty.

So, when making a FPS Game, you always want to keep weapon balance in your back pocket at all times. What could easily balance this is to block snipers the one hit kill ability unless it's a head shot. Generally, it's a good idea to avoid any one hit kill weapons in general, unless it's freaking impossible to live from (IE: RPG, Tank Shell, Bomb, ect.).

Next up, it's important when building a good gun balance, to incorporate the element of random into your shots, so unless you've got a precision weapon in your hands, you should make use of the projectileSpread option in the weapon image to make shots precise up close, but taper off in accuracy with distance. Trust me, nobody wants an automatic super-duper sniper rifle with 0.0000 projectile spread firing 5 rounds per second. You can also add recoil to your gun to fix this problem, Torque 3D has a "recoil" field in the Fire state of the weapon image, make use of it.

So, that's the end of my little "rant" about gun balance, let's now talk about the actual design process and how you should go about making your weapons for your shooter game.

The first thing you need to do, is have a nice sit down session, maybe with some coffee, or whatever other computer programming related beverage you want to indulge yourself in. Next, ask yourself this question: "What weapon suits my genre?". This one's big, unless you're making a game titled "BL3ND 0F DUTY: TIME & SPACE DIST0RT10N", you'll want guns that fit the time period, if not slightly ahead and behind of it. For example, in a modern shooter game, I'm not going to say, alright folks, we've got Star Wars Blasters and Lightsabers for your weapons, go have fun. Likewise, I'm not going to say, "In a world where nukes threaten our existence, your elite team has the greatest of all in medieval gear you could possibly ask for, we've got honor with swords and bows, and… " I'll just stop that there. If you're making a modern shooter game, have modern weapons. Use guns that people are familiar with. AK-74 (Or AK-47 for you terrorist lovers), Tar-21, F-2000, RPG-7, ect. And then jump out of the normal with guns that may "seem" and "feel" familiar, but with your own custom touch. For example, you could take a SCAR-H and instead, give it the handle and barrel of the M4A1, and voila, you've got yourself a brand new assault rifle.

If your game calls for weapons outside the box, then it's time to get inventive with names. For example, In my FPS Game (Tactical Uprising), the timeline is slightly ahead

of the current time, so while blasters and laser swords might not be the best choice, pulse and plasma weapons do fit the description. I've got a XVM Pulse Rifle and a ZX44 Pulse SMG in my arsenal. Don't be afraid to jump out of the ordinary in order to obtain the weapons you want to create, especially with naming, players absolutely fall in love with certain weapons, and if the name is even more interesting, they'll have an even better reason to like it.

Next, if you're making modern weapons, you can jump into the modelling process with reference images. If you don't have a reference image, you can try modelling by hand (which is hard), or you can draw yourself a nice sketch and then scan it into your computer and model from there.

Next up is behaviors of weapons. Now, when you make weapons you're generally going to have a few classes of weapons, which are common to all games and they are as follows:

The first is you high powered, low ammo guns, such as your explosives, sniper rifles, and shotguns. These guns will usually be extremely effective in one situation and that one situation alone, for example a shotgun will be extremely effective in a "in your face" showdown, whereas you'll just want to run away when the other guy is at a longer range with an assault rifle. A great way to see this is the class of high risk, high reward weaponry when effectively used.

Next up is the standard weaponry, such as sub-machine guns, assault rifles, and maybe a few other guns. Your standard weapons will be all around usage guns that are applicable in almost every showdown. They will work in almost all situations except when the high powered gun has the edge, in which case they will be seen as reaping the high reward of having the high advantage.

Last is your low risk weaponry, these guns will output light to moderate damage but either have high ammo counts and/or high firing speeds. These guns (such as Light Machine Guns) will work in pretty much any situation, but unless the person firing the weapon gets the first shot off, they will likely lose the gun fight.

Now, you don't need to place my examples in these categories, but the way your game is set up will generally weed these out with time anyways, you'll notice the players who want to take a more competitive edge use your high risk guns often in combination with standard weapons while the more casual player will want to try every possible gun with every combination just to have fun in the game out. And this is where you, the developer can enter another aspect of play.

Weapon Balancing after release is very important, from my FPS Design tutorial I explained how you will never get it right, no matter how hard you try, but you can only

make it better with time. Make sure you incorporate some kind of patching solution to your game to be able to quickly adjust the values of your weapons to balance them out. Give certain low powered guns a high power flip every once in a while, or knock down a high power gun into the standard class to make it fell right to all of the players in your game. And whatever you do, do not, I repeat, DO NOT, do what Call of Duty does by completely ignoring the community and just catering to your own style or the style of the "select" few players in your game who just might be more well known. Listen to all feedback, when someone complains, it's not just because they lost the game, or got decimated by someone with a better playstyle, sometimes, some weapon setups are truly overpowered, or maybe a setup that should be good is underpowered. Make sure you go through each request and piece of community feedback and perform the proper adjustments. And after you adjust a gun, ask for additional feedback from your community, make sure you got it right by checking in with the majority of your players. The more you listen, the more your community will grow and thrive. In time, you can balance all of your weapons just based on feedback alone.

So, now, you've got a name for the gun, you've figured out where it belongs in the classification, next up is how you want to present the weapon. Weapon presentation is almost as important as weapon balance. Some guns will just be handed to the players, while other guns, you can give them very flashy intros, or show off the power to the player to make them see that "this is the gun they want", and it comes down to a few things. Good modelling/texturing is one, Good sounds are the next element, and probably the big one is the visual effects (particles) of the gun when it fires off. If I give a player a giant plasma cannon with the big red "DO NOT PRESS" button on it, this thing better do something very awesome, or you're going to have a very disappointed player, another thing, unless your game is called "0H MY SW33T FPS L055", don't overdo the particles. The last thing you need is an artificial FPS killing fog created by particle effect spam.

So when it all comes down to making good weapons, make sure that:

1: It fits the Genre of your game.

2: The weapon idea is balanced compared to the other guns.

3: Your weapon's concept is new, cool, and is something a player will love.

4: It fits nicely in the three-type perspective model.

5: Always take community feedback regarding individual weapons seriously.

6: Make sure the gun fits it's "look" factor.

Getting all of these elements down will help you in your quest to bring down the FPS Giants, and they will make your game much more likable and enjoyable to players of all skill levels.

Part 4: Tips & Tricks

The last part of this Weapon Design tutorial is going to be my little tips & tricks section where I'll show you how to do some "less common" things with weapons. Consider this the "Robert Fritzen's Torque Weapon Gems Section".

First thing I'll show you my code for grabbing a target position using a weapon.

```
$ArtilleryMask = $TypeMasks::TerrainObjectType | $TypeMasks::StaticObjectType |
$TypeMasks::PlayerObjectType | $TypeMasks::VehicleObjectType |  $TypeMasks::TurretObjectType;
function FetchTargetPosition(%source) {
   %pos       = %source.getMuzzlePosition($WeaponSlot);
   %vec       = %source.getMuzzleVector($WeaponSlot);
   %targetpos  = vectoradd(%pos,vectorscale(%vec,99999));
   %targetPos  = getWords(containerraycast(%pos,%targetpos,$ArtilleryMask, %source), 1, 3);
   return %targetPos;
}
```

This function will allow you to pull a target position from your weapon's muzzle vector. You can adjust the maximum allowed distance by changing 99999 to something else if you'd like to cap the distance here. Let's say I want to call in an artillery strike on the target position, well using the above function I can write a weapon OnFire call that looks like this:

```
function myWeaponImage::onFire(%this, %obj, %slot) {
   parent::onFire(%this, %obj, %slot);
   //Fetch Position
   %target = FetchTargetPosition(%obj);
   DoArtilleryStrike(%obj, %target);
}
```

We pass `%obj` to this function call to allow us to define the projectile source as the player who called in the strike. When you spawn the new projectile you'll use `%projectile.sourceObject = %obj;` to do this. This for example is my code from Tribes 2 to call in a wave of 8 artillery bombs:

```
function DoShadowArtillery(%source, %targetPosition) {
   for(%i = 0; %i < 8; %i++) {
      %pos1 = vectorAdd(%targetPosition, "0 0 200");
      %pos2 = vectorAdd(%pos1, GetRandomPosition(15, 1));
      %final = vectorAdd(%pos2, "0 0 "@%i * 25@"");
      %p = new (GrenadeProjectile)() {
         dataBlock        = ShadowBombShot;
         initialDirection = "0 0 -1";
         initialPosition  = %final;
         damageFactor     = 1;
      };
```

```
      MissionCleanup.add(%p);
      %p.sourceObject = %source; //hacky way of spawning airborne projectiles
   }
}
```

Obviously I've got a few more "custom" things in there, such as getRandomPosition(), which I've pasted right below, and Tribes 2's GrenadeProjectile, which is simply the Projectile with isBallistic = true;

Here is my code for generating random positions using a few parameters:

```
function GetRandomPosition(%mult,%nz) {
   %x = getRandom()*%mult;
   %y = getRandom()*%mult;
   %z = getRandom()*%mult;

   %rndx = getrandom(0,1);
   %rndy = getrandom(0,1);
   %rndz = getrandom(0,1);

   if(%nz)
      %z = 5;

   if (%rndx == 1){
      %negx = -1;
   }
   if (%rndx == 0){
      %negx = 1;
   }
   if (%rndy == 1){
      %negy = -1;
   }
   if (%rndy == 0){
      %negy = 1;
   }
   if (%rndz == 1){
      %negz = -1;
   }
   if (%rndz == 0){
      %negz = 1;
   }

   %rand = %negx * %x SPC %negy * %y SPC %Negz * %z;
   return %rand;
}
```

With this you can randomize positions based on a few positions and a single call to vectorAdd().

Something you will learn over time, is that the subject you probably hated the most throughout your childhood (math), will actually prove to be your most powerful weapon in terms of game development. I'll be honest, I actually learned more about vectors in

developing games than I ever learned in my calculus course, and I actually walked into that class knowing about 90% what was taught in vectors just from Torque alone.
Next up, I want you to learn how to apply for() loops in a new way. So, you've probably used a few of these by now to loop over a task 'x' amount of times when you know 'x'. So, why not use this to help us do something cool with math.

```
for(%x = 0; %x < 360 %x += 30) {
   %y = (mCeil(%x / 90) * 90) - %x;
}
```

With this for loop you can spray projectiles around the unit circle with some cosine and sine calls in the vector calculation, allowing for some interestingly evil weapon setups. For example, let's say I have a surface bomb that fires out toxic darts in a circle pattern when it explodes, well, there you go.

Speaking of "Toxic" weapons, how about we add a "burn" debuff, which is actually quite simple:

```
function burn(%object, %src) {
   if(!isObject(%object) || %object.getState() $= "dead") {
      return;
   }
   %object.burnCounter++;
   if(%object.burnCounter >= 20) {
      %object.burnCounter = 0; //reset…
      return;
   }
   //Damage Code here, maybe some burn effect too…
   Schedule(100, 0, burn, %object, %src);
}
```

And it's as easy as that. To apply the burn, say when a fire bolt hits the player, we do this:

```
function fireBolt::onCollision(%this,%obj,%col,%fade,%pos,%normal) {
   //other code…
   %col.burnCounter = 0;
   burn(%col, %obj);
}
```

And watch the guy scream as he wonders why his health is slowly going down for a little while. So, we've got some basic stuff down, how about some more interesting segments of code. I wrote a mod for Tribes 2 called Powers Mod, where I had players level up and unlock evil little death spells to fire at each other, one of those little nasty ones launched a stream of explosions in a chain pattern across the ground. And here's how I accomplished that:

```
function createFissurePulseOne(%sourceObject) {
   if(!%sourceObject.isAlive()) {
      return;
```

```
   }
   %mp = %sourceObject.getMuzzlePoint(0);
   %vec = %sourceObject.getMuzzleVector(0);
   // This projectile is the pointer for the explosions on the surface.
   %TPos = getWords(%mp, 0, 1) SPC (getTerrainHeight(%mp) + 3000);
   //
   %p = new SeekerProjectile() {
      datablock = FissurePointer;
      InitialPosition = %TPos;
      InitialDirection = %vec;
   };
   %p.sourceObject = %sourceObject;
   MissionCleanup.add(%p);
   //
   schedule(1500, 0, "FissureBurstLoop", %sourceObject, %p, 0);
}

function FissureBurstLoop(%src, %proj, %count) {
   if(!%src.isAlive()) {
      if(isObject(%proj)) {
         %proj.delete();
         return;
      }
      return;
   }
   if(%count > 20) {
      if(isObject(%proj)) {
         %proj.delete();
         return;
      }
      return;
   }
   else {
      %count++;
      //
      %grd = getTerrainHeight(%proj.getPosition());
      %explo = getWords(%proj.getPosition(), 0, 1) SPC %grd;
      //
      %fisBurst = new LinearProjectile() {
         datablock = FissureBurstExplosionProjectile;
         InitialPosition = vectorAdd(%explo, "0 0 1");
         InitialDirection = "0 0 -1";
      };
      %fisBurst.sourceObject = %src;
      MissionCleanup.add(%fisBurst);
      //
      schedule(500, 0, "FissureBurstLoop", %src, %proj, %count);
   }
}
```

How it works is you spawn a projectile way up in the sky to serve as the “pointer”, you can define the speed of this projectile to define the speed of progression and basically as the projectile moved in the sky, the location of the explosions at the surface would appear to “move” along the ground in a direction. Sometimes you’ve got to think of things outside the box to accomplish what you want them to do.

Now, let's say you've got a gun that creates an explosive, and you want everyone hit by said explosive to feel some kind of effect. That too, is very easily done:

```
function ConcGunGrenade::onExplode(%data, %proj, %pos, %mod) {
   Parent::OnExplode(%data, %proj, %pos, %mod);
   %TargetSearchMask = $TypeMasks::PlayerObjectType;
   InitContainerRadiusSearch(%pos, %data.damageRadius, %TargetSearchMask);
   while ((%potentialTarget = ContainerSearchNext()) != 0) {
      if(%potentialTarget.getState() !$= "dead") {
         if(((%potentialTarget.client.team == %proj.sourceObject.client.team) && $TeamDamage) ||
%potentialTarget.client.team != %proj.sourceObject.client.team) {
            //Do Stuff to %potentialTarget
         }
      }
   }
}
```

How this one works is it scans over the projectile's damage radius for players who are not on the same team, at which point you can apply whatever kind of effect you want to apply on the player. Just remember that all of these code bits are written on Tribes 2 and therefore you'll need to edit them for your own needs (AFPSK actually has a %object.getTeam() method to replace the %object.team way of things).

All of these little code bits are only little samples of the kind of things you can accomplish with some time and thought. And as long as you have the imagination to create interesting weapons of the likes players have never seen before, the spark of First Person Shooter Game will always remain closely behind.

I hope you've enjoyed this tutorial, and as always, if you need any help, feel free to ask us any questions!