Programming for Artists and Designers

Tutorial 6: Introduction to Weapons

Copy Right:
This tutorial uses ideas and code from the following people:
People @ The unreal wiki – see links below
Jeff Giles @ Sega 500

Aims: To introduce you to weapon modding

Introduction:
Weapons are a huge area, just like many things in unreal. Again it will be impossible for me to cover, in the time, every aspect of making a weapon, although I will do my best to give a broad introduction.
The solution to this is to read. The majority of the information, if not all, is available online so surf and read what you find.

Note:
When looking around the web be careful that any sources you read are for UT2K3 and not for the original UT. This is because weapon functionality has completely changed since that version.
See online about using 3DSMax to import weapons.

Homework:
Read lesson 19 http://gamestudies.cdis.org/~jgiles/Lessons.htm

States

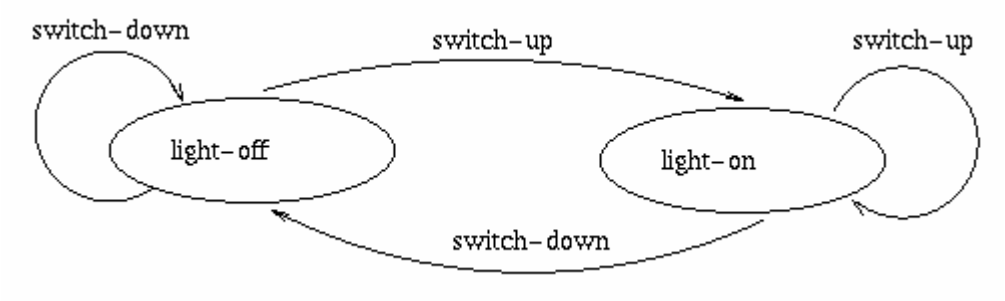
Dictionary Definition: A state may be considered to be a point in some space of all possible states. A simple example is a light, which is either on or off. A complex example is the electrical activation in a human brain while solving a problem.

In computing and related fields, states, as in the light example, are often modelled as being discrete (rather than continuous) and the transition from one state to another is considered to be instantaneous. Another (related) property of a system is the number of possible states it may exhibit. This may be finite or infinite. A common model for a system with a finite number of discrete state is a finite state machine.

Finite State Machine: An abstract machine consisting of a set of states (including the initial state), a set of input events, a set of output events and a state transition function. The function takes the current state and an input event and returns the new set of output events and the next state.

The light example is either on or off, and it starts in one of these states. The state it starts in is the initial state. The cause of transition in state is known as an event.

These states are often planned out in a state transition or bubble diagram. These diagrams depict the state, transitions and events.



As far as unreal script is concerned a state is a section of code that is executed if the class is in that state.
The language reference contains more on this.

States, transitions and there diagrams are used for many thing it is therefore a massive subject area beyond the scope of the course. Here is a look at some of the more OOP areas though…. if you're interested that is
http://www.cs.unc.edu/~stotts/COMP145/CRC/state.html

What do States Have to do With Big Guns
Well weapons have states, fire, alt-fire, reload.

How Firing Weapons in UT2003 Works
Everything begins on the PlayerController Fire exec function. An exec function is a function that can be bound to a key by a player, so you won't find it directly called in most UnrealScript. This is because the engine handles the event handling natively.

```
exec function Fire( optional float F )
{
    if ( Level.Pauser == PlayerReplicationInfo )
    {
        SetPause(false);
        return;
    }
    Pawn.Fire(F);
}
```

If the level is not in pause mode, call the Pawn's Fire function. If the level is paused and the person who paused it is the one who's firing, it unpauses the level but does not fire the weapon.
Pawn.Fire

```
function Fire( optional float F )
{
    if( Weapon!=None )
        Weapon.Fire(F);
```

```
}
```

If the pawn has a weapon, call the weapon Fire function - Weapon.Fire

```
simulated function Fire(float F)
{
     // Weapon fire code in here
}
```

WeaponFire is responsible for creating the visual effects and applying damage.

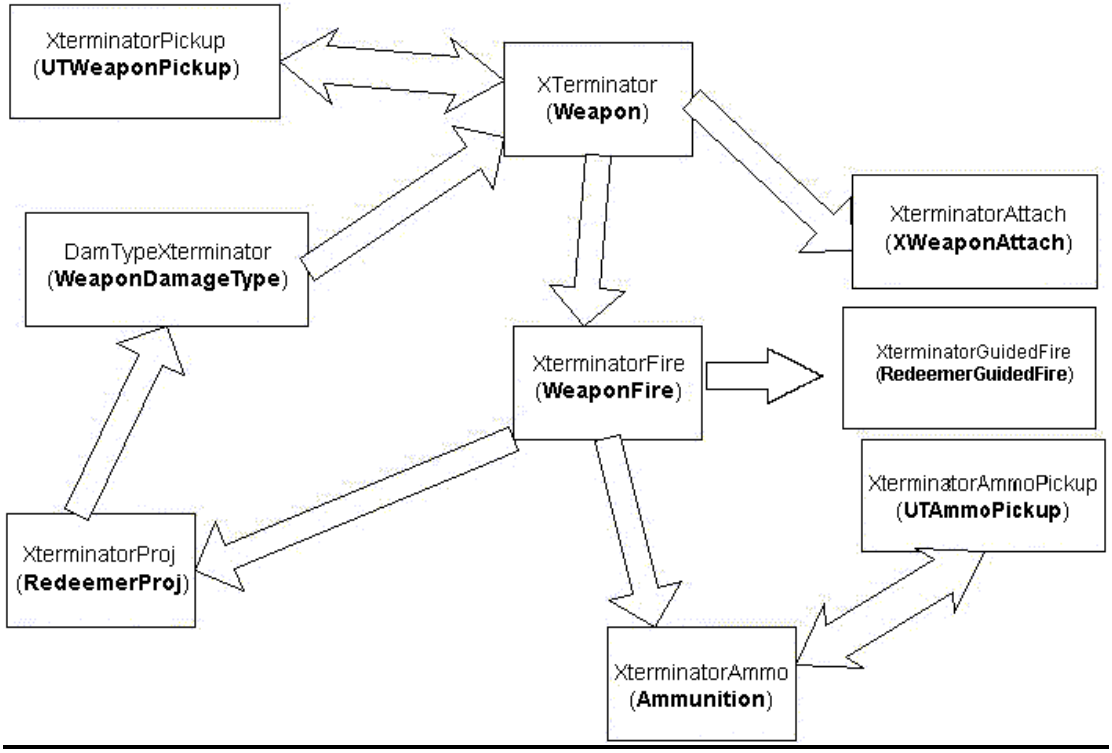
There are two main types of weapon fire type:
Projectile and Instant fire

Projectile fire is easier to handle so we are going to cover a projectile weapon this week.

How Many Classes?

To radically alter the functionality of a weapon you don't really need to touch many classes.
To get a unique weapon functioning correctly you need to alter a minimum of 8 classes.

What are all these Classes?



These classes are the ones we need to create a weapon

Weapon - Inventory and Pickup
Ammo - Inventory and Pickup
Projectile – Type, Fire Class and Damage Type
Attachment Class

A Complex Weapon

We will now look at making a weapon.
The weapon will essentially be a variant of the redeemer, called the Xterminator. This wouldn't make a good mod, but makes a fun tutorial.
A looking at the parent classes of the files we need to inherit I reckon the **weapon** class is the place to start.
If you load up the weapon class you will find that it is massive, all that functionality is a necessary requirement of weapons in UT2K3.

The Xterminator Class – Ok so we want the functionality of the redeemer, the best way is to define our class like this:

```
//=============================================================
// Looks like a Rocket Launcher Hits like a Redeemer
//=============================================================
class Xterminator extends Redeemer
    config(user);
```

Another way is have Xterminator extend the standard weapon class. This means that any of the functions defined in the Redeemer are not inherited. This is not always a big loss.

Change the following default properties

```
FireModeClass(0)=Class'XterminatorFire'
FireModeClass(1)=Class'XterminatorGuidedFire'

PickupClass=Class'XterminatorPickup'

AttachmentClass=Class'XterminatorAttachment'

IconMaterial=Texture'InterfaceContent.HUD.SkinA'
IconCoords=(X1=745,X2=866,Y2=88)
ItemName="Xterminator"
Mesh=SkeletalMesh'Weapons.RocketLauncher_1st'
DrawScale=1.500000 // make it big

UV2Texture=Shader'XGameShaders.WeaponShaders.WeaponEnvSha
der'
```

If you wondered what does config(user) means, here is your answer:

config or config(name)

This class supports saving its config properties to a configuration file. By default this is the Game Ini File (the configuration file that has the same name as the executable file of the game, e.g. UnrealTournament.ini or UT2003.ini). Using config(name) overrides the default config file name for this class. There are two special names that can be used here: System uses the game ini file, User uses the User.ini. Both names can be mapped to other files via the games -ini=... and -userini=... [command line parameter]?s.

The Fire Class – This bit is easy to code as we are using the Redeemers functionality. Open up the RedeemerFire file, found in xWeapons. We are only inheriting one function from the RedeemerFire class and all the other from the ProjectileFire class.
Anyway, all we need to do is to add this to our file.

```
class XterminatorFire extends RedeemerFire;

defaultproperties
{
     AmmoClass=Class'XterminatorAmmo'
     ProjectileClass=Class'XterminatorProjectile'
}
```

All we are effectively doing is defining what we shoot

The Alt Fire Class – What we need to do now is to copy the RedeemerGuidedFire file and rename it after our class:

```
class XterminatorGuidedFire extends XterminatorFire;
```

We only need to alter one bit as well:

```
HurtRadius(500, 400, class'DamTypeXterminator', 100000,
Instigator.Location);
```

The reason for doing this is that we want the guided fire of the redeemer with our new explosive radius.

The Ammunition Class – This bit is vastly different from the Redeemer ammo class. The reason being all we would get would be a single shot redeemer as normal. What we want is multiple shots.
Our Ammo class looks more like the rocket launcher ammo class.

```
class XterminatorAmmo extends Ammunition;

#EXEC OBJ LOAD FILE=InterfaceContent.utx

defaultproperties
{
     MaxAmmo=30
```

```
     InitialAmount=15
     PickupClass=Class'XterminatorAmmoPickup'
     IconMaterial=Texture'InterfaceContent.HUD.SkinA'
     IconCoords=(X1=744,X2=645,Y2=74)
     ItemName="Xterminator Ammo"
}
```

This basically tells the weapon which ammo type to use.

The Ammo Pickup – Those of you who play the game may realise that there is no Redeemer Ammo, it's a one shot weapon. That's why we've made our own Ammo Pickup based around the rocket launcher ammo pickup, infact they even look the same as I've used the same mesh

```
class XterminatorAmmoPickup extends UTAmmoPickup;

#exec OBJ LOAD FILE=PickupSounds.uax

defaultproperties
{
     AmmoAmount=15
     MaxDesireability=0.500000

     InventoryType=Class'XterminatorAmmo'

     PickupMessage="You picked up some Xterminator ammo"
     PickupSound=Sound'PickupSounds.FlakAmmoPickup'
     PickupForce="FlakAmmoPickup"

     DrawType=DT_StaticMesh

StaticMesh=StaticMesh'WeaponStaticMesh.RocketAmmoPickup'
     CollisionHeight=13.500000
}
```

The Projectile Class – Again we are pinching the functionality of the redeemer so all we want to do is alter some defaults.

```
//=============================================================
// One Big Mutha of a Rocket.
//=============================================================
class XterminatorProjectile extends RedeemerProjectile;

defaultproperties
{

     DamageRadius=2500.000000 //Make the explosion bigger
     MyDamageType=Class'DamTypeXterminator'
}
```

The Damage Type – In UT2003 DamageTypes are abstract classes which hold information like e.g. the corresponding death messages and hit effects.

```
class DamTypeXterminator extends WeaponDamageType
                         abstract;

defaultproperties
{
     WeaponClass=Class'Xterminator'

     DeathString="%o was XTERMINATED!"
     FemaleSuicide="%o Suicide XTERMINATION!"
     MaleSuicide="%o Suicide XTERMINATION!"

     bArmorStops=False
     bSuperWeapon=False //of course it isn’t  ;)
     bKUseOwnDeathVel=True
     KDeathVel=600.000000
     KDeathUpKick=600.000000
}
```

The Weapon Pickup – How the item appears in the world before pickup & what item it spawns into the players inventory

Copy the redeemer pickup but add these bits:

```
//=============================================================
// XterminatorPickup
//=============================================================
class XterminatorPickup extends UTWeaponPickup;

defaultproperties
{
     MaxDesireability=0.900000
     InventoryType=Class'Xterminator'
     RespawnTime=60.000000
     PickupMessage="Check it out, its proper Bo!"

     PickupSound=Sound'PickupSounds.FlakCannonPickup'
     PickupForce="FlakCannonPickup"

     DrawType=DT_StaticMesh

StaticMesh=StaticMesh'WeaponStaticMesh.RocketLauncherPick
up'
     DrawScale=0.900000
}
```

The Attachment Class – This handles the visual aspects of the weapon in third person.

The functions used the most are:
UpdateHit - Used to update the properties of the attachment so the client side can correctly spawn effects.
ThirdPersonEffects - This function handles the generation of muzzle flashes, animation, tracers, etc on the client side. Super.ThirdPersonEffects should always be called.

For this mod we have copied the redeemer attachment class and just changed two lines in the default properties:

```
DefaultProperties
{
   //how the weapon appears in 3rd person
   Mesh=SkeletalMesh'Weapons.RocketLauncher_3rd'
   DrawScale=1.500000
}
```

The int File:
Add this line to the int file:

```
Object=(Class=Class,MetaClass=Engine.Weapon,Name=MyPackag
e.Xterminator,Description="The Xterminator")
```

Making sure it Works
Load up a game and then bring down the console using the ‘ key and then type the following:

Summon CVGpackage.XterminatorPickup

Inventory and the Astute Thinking
The more astute of you may have thought, hey if there isn’t an ammo pickup or this weapon is a brand new one how does it become part of our inventory. Well this is one of the clever bits to UT2K3. When we collect an item it spawns a different item in our inventory for use by the player.

Inventory is the parent class of all actors that can be carried by other actors. Inventory items are placed in the holding actor's inventory chain, a linked list of inventory actors. Each inventory class knows what pickup can spawn it (its PickupClass). When tossed out (using the DropFrom() function), inventory items replace themselves with an actor of their Pickup class.

In our case XterminatorAmmoPickup creates the XterminatorAmmo inventory item and the XterminatorAmmo knows what item it creates.
This is the reason for the double arrow in the diagram.

A Simple Weapon Mutator
See if you can program a mutator where either one type of weapon or all weapons are replaced with the Xterminator.

Important Links:
http://wiki.beyondunreal.com/wiki/Weapon

http://wiki.beyondunreal.com/wiki/InstantFire

http://wiki.beyondunreal.com/wiki/DamageType