Programming for Artists and Designers

Tutorial 2: Advanced Mutator

Aims:
This aims to get you programming a more advanced mutator; it builds upon what you started learning last week.
It also aims to advance your knowledge of unreal script.

Note:
As you have a go at writing each piece of code read the notes that go with it, they not only describe what your doing but also certain describe what each line means to unreal script.
You don't need to load up unreal editor for this.
It builds on last week and so assumes you remember everything about last weeks tutorial.

You may also see bits like this jotted around:

Inheritance: UT2003 :: Actor >> Info >> Mutator (Package: Engine)

This is to inform you of the inheritance order of certain classes.

Modifying Last Weeks Package:
As you're not assured of getting the same computer each week it is easier if everyone gives the package used during tutorial the same name. If we from now one call it CVGpackage – this will make subsequent weeks scripting easier.

Look in the INI file at the edit packages line and the bottom Editpackages = line should have something like "EditPackages = /whatevertheycalledtherepackage/".
Change this line to "EditPackages = CVGpackage"

Remember we want the name adding to the first EditPackages line, or both, in the ini file not the second line. Adding it to the second and not the first will cause errors when it comes to opening the unreal editor and will result in our package not being compiled.

As we are going to be creating another mutator and re-compiling we need to either rename the .int file, to that of our package name or add a new one

We recompiling if something doesn't work we need to delete .u file of the package this is because when we recompile if there is an .u file present UCC won't recompile it.

A Word on INI Files:
The Unreal Engine based games make use of one or more INI files to hold configuration information for the classes used within the application running.

Some mods also use their own configuration file to avoid cluttering up the main program INI file and making their mod easier to install and uninstall.

The INI file is broken into sections each with its own header. Each section within the INI file corresponds to the configuration of a class. The format of the section header is typically [package.classname] but not always.

Start Coding:
At the end of this tutorial you should have a mutator that regenerates your adrenaline.

Here is the class is full:

```
//=============================================================
// MutBooster - Boosts Adrenaline
//=============================================================

class MutBooster extends Mutator;

#exec OBJ LOAD File=MutatorArt.utx

var() int AdrenalineIncrease;

// Don’t call Actor PreBeginPlay() for Mutator
event PreBeginPlay()
{
    SetTimer(1.0,true);
}

//=============================================================

function Timer()
{
    local Controller C;

    for (C = Level.ControllerList; C != None; C =
C.NextController)
    {
         if (C.Pawn != None && C.Adrenaline < 100 )
        {
            C.Adrenaline += AdrenalineIncrease;
        }
    }
}

//=============================================================

defaultproperties
{
     AdrenalineIncrease=2
     GroupName="Booster"
     FriendlyName="Booster"
     Description="Boosts adrenaline"
}
```

Breaking It Down:
This bit may seem pretty boring to you, but it is stuff you need to know if you want to code in unrealscript, or even pass the module.

```
//=============================================================
// MutBooster - Boosts Adrenaline
//=============================================================
```

- This bit should hopefully be self explanatory - This is commenting and is here to tell anyone who opens up the folder what it does and where its from.

```
class MutBooster extends Mutator;
```

- This says that MutBooster is derived from or is a child of the class mutator.
- I order to make our specific mutator work; we simply extend Mutator's functionality by modifying or adding to those traits. This is true for game types and weapons – something for another time.
- The class declaration. Each class "expands" (derives from) one parent class, and each class belongs to a "package", a collection of objects that are distributed together. All functions and variables belong to a class, and are only accessible through an actor that belongs to that class. There are no system-wide global functions or variables.

```
#exec OBJ LOAD File=MutatorArt.utx
```

- Exec directives like this one tell the engine to interact with real files, such as models and textures. This one is used because it was in MutRegen and so seemed appropriate. They become more important when you start trying to retexture existing models or texturing you won models.
- They execute console commands at compile time

```
var() int AdrenalineIncrease;
```

- This declares an integer variable of type int, it holds a 32 bit integer value
- It is also what is known as an instance variable. These apply to an entire object and appear immediately after the class declarations

```
// Don’t call Actor PreBeginPlay() for Mutator
event PreBeginPlay()
{
    SetTimer(1.0,true);
}
```

- This redefines the event PreBeginPlay to start a timer.

- It is called before play starts. This function is used to initialize variables and objects that need to be set before the game begins
- SetTimer has two variables - the first is the interval time for the timer, or how long to wait. The second is a boolean which sets the timer to repeat or not. Here, we set the timer to repeat about every second. It looks like this in its base form –

SetTimer(float NewTimerRate, bool bLoop) [native, final] and it causes *Timer()* events every *NewTimerRate* seconds.

```
//=============================================================

function Timer()
{
    local Controller C;

    for (C = Level.ControllerList; C != None; C =
C.NextController)
    {
         if (C.Pawn != None && C.Adrenaline < 100 )
        {
            C.Adrenaline += AdrenalineIncrease;
        }
    }
}

//=============================================================
```

- The function Timer is what is actually called when the timer is triggered.
- This creates a local variable to hold a Controller. Controllers are new to UT2003, but could be easily described as the code which actually controls the pawn (player or bot) in the game. Pawns and Controllers work tightly together, allowing us to modify distinct parts of them individually. Here, it runs through the current list of controllers (Level.ControllerList) and if that controller doesn’t have a certain amount of adrenaline, it increases it.
- The controller class is a handle to a pawn. Pawns are the physical (graphical) player characters; the controller is the controlling entity moving it around.

Inheritance: UT2003 :: Object >> Actor >> Controller (Package: Engine)

- The "C.Pawn != None" test in the innermost if statement means that it won't perform the logic if that object isn't there. It is very similar to the NULL statement in other programming languages.
- It is a vital part of scripting Unreal, because not catching instances where an object doesn't exist can not only break your code, but also slow down the entire game as the game logs the "Accessed None" error.

- You should be able to tell me what the rest means

```
//=============================================================

defaultproperties
{
     AdrenalineIncrease=2
     GroupName="Booster"
     FriendlyName="Booster"
     Description="Boosts adrenaline"
}
```

- This is a special part of the UnrealScript Class syntax. It's block of script at the end of a class that declares default values for class properties. This block cannot contain any executable code.
- Default properties are inherited by subclasses, and may in fact be overridden several times as you go down the class tree. The most "recent" (lowest subclass that has a default property for a class property) takes precedence.
- In this case it defines some of the basic variables for the mutator.
- AdrenalineIncrease here is used in the timer to determine the amount of adrenaline to increase, and if this mutator were configurable - that number could be changed to alter the mutator's effects without having to alter any code. The other three define the mutator for the game. The "GroupName" is the group the mutator belongs to. There can only be one mutator of a group brought into the game at one time. "FriendlyName" is the title placed into the mutator list and "Description" is what will go into the description block when the mutator is highlighted.

Finally, Altering the INT File:
The last bit before your mutator works is to alter the INT file; you should remember how to do this from last week.
Have a go then, the syntax is almost identical.

- INT files are one of the Unreal Engine's file formats.
- These files are used by the Unreal engine for reference. Instead of searching all the packages made accessible to it to find the resources it's looking for, it simply reads the lists of resources contained in these files. In addition, they are used for language localization and that's the origin of the file extension .int for "international" as well.
- You can do so much with INT files that they probably need their own tutorial

Your Very Own Mutator:

Once you reach this stage the next step is walking…try and create your own mutator – first come up with an idea, but don't spend to long on it.
Ideas include low gravity that activates every 30 seconds for 5 seconds, a shield regenerator or a health decreaser, your health gets lower every few seconds.
Look at other mutators to get an idea of how to do something, such as the low grav mutator.