Programming for Artists and Designers Assessment 1

This is the first of two assessments that contribute to your mark for the Programming for Artists and Designers module. Each assessment counts for 50% of the total marks for the module.
You are reminded that the work that you submit for this assessment must be entirely your own. If you copy some or all of another student's work, or write any, or all, of your program with the assistance of anyone else, you will be subject to disciplinary procedures. These may result in you being awarded zero marks for the whole assessment, or other, more severe penalties. If you allow another student to copy some or all of your work, you will be subject to the same disciplinary procedures and penalties as if you had copied the other student's work.
Never store your program code on the PCs' hard drives, as this would allow other students to copy your work. If you print a copy of the make sure you collect it immediately.
Stage 1 – The proposal should be submitted to the Music, Media and Performance School office, by 12:00 p.m. on Friday 6th March 2008. The file should have your name clearly displayed on it. It is extra important you get this submitted on time.
Stage 2 – You must hand in your program **code files and directory structure** (i.e. the package folder and its sub-directories: classes and textures), preferably on a CD but a floppy disc will also do. The code files also include your .int file. You should also submit **hard copies** (print outs) of your code to the Music, Media and Performance School office, by 12.00 p.m. on Friday, 11th April 2008. The program code must contain a comment giving your name and a statement that the program is all your own work. You may be required to explain what your program code does, in an oral interview with me, or with another member of staff.

The Assessment

The assessment is broken up in to 2 parts. One part involves you following my code to produce a working modification; the other part involves you coming up with your own code as a solution to a problem.

Part 1

The first part involves completing and submitting the completed tutorial 7 exercise. This tutorial looks at coding a more complex HUD.
See tutorial 7, it can be downloaded from:
http://www.activehelix.com/progart.html

Part 2

The second and final part of the exercise involves you completing your own mod by completing the following stages.

You have been approached by team of professional modders who wish you to write a basic new game type for them. They wish the game type to include its own HUD and mutator.

<u>Stage 1 - The Proposal:</u>
A written proposal should be completed; it should be approximately one to two sides of A4, typed.
It should be submitted by the deadline indicated above.
It should cover what you intend to do for the gametype and mutator, roughly a paragraph each. It should also cover any information relevant to the actual completion of the project that you have discovered through research. It is important to look at the links provided online.
This is to save you time and effort from proposing to do anything that will take a ridiculous amount of time and effort.

<u>Stage 2 - Coding:</u>
The projects code is broken down into two parts; each one should have its own code file. Along with the code files I wish the .int file to also be submitted.

Game Type:
You are free to choose your own game type, the only stipulation being that it is original and not copied. The game type doesn't have be as complicated as capture-the-flag but can be as simple as the TopKiller game type covered in tutorial 3.

The game must not involve the importing of additional graphics.

Mutator:
The mutator must be something that has quite an impact on the game, such as occasional low gravity or regenerating adrenaline. Again it doesn't have been exceptionally complex.
The mutator must not involve the importing of additional graphics.

<u>Building on what has come before</u>
Much of your code will use functions and ideas that are already covered in the games script files. Use these get an idea of how the functions you are altering look and how they implement them. Example is if you were coding a mutator for low gravity where low gravity was active only some of the time, or at random times then you would at the already available low gravity mutator.

Massive Code Base
Remember unreal has a massive code base and it is therefore impossible for us to cover every possible function in class, use online materials provided in the links section online encounter a function that you don't recognise. If you are still stuck get in touch with me and ask.

Commenting
You are marked on your comments; you are therefore reminded that your code should be more than adequately commented.
See the resource file online for an example of the commenting you are expected to give.