

## The Role and Importance of the Audio Programmer In Computer Game Development

### i / Introduction

In the world of modern media based entertainment, consumers are used to and expect a very high fidelity product. This is true whether it be audio, visual, written or a hybrid such as modern films. While it is difficult to create this using a linear based delivery method such as film, (where circumstances and elements are static), it becomes tenfold when employing an interactive and non-linear technique. In such situations, aspects of the production are defined in real time by the user, and extra systems are needed to be put in place in order to create the same high quality product.

In computer game audio the audio programmer is that system.

### ii / A Brief Example

I would like to take a moment to give the reader a quick example of the fundamental difference between linear and non-linear, interactive entertainment.

You are in a dark corridor with three doors; one, two and three. In linear entertainment (such as film) you would have no preference which door you go through first, at what speed, or any other choice, you are a spectator, along for the ride. In contrast, with non-linear interactive entertainment you can go through any door you want first, any speed, you can even look around for the light switch first, you are in control. Whilst this seems a very obvious difference it brings around the need for two completely different production systems. In the linear example every sound can be “spotted” and placed meticulously because it will **always** be in that particular place and time. Meanwhile in the non-linear example that is not the case, the audio has to be mapped to the user’s reactions, not the other way round.

### iii / What is an Audio Programmer?

The role of the audio programmer can be very varied and require many differing skill sets. This taken into account, it can be broken down into four broader main sections; specification, design, implementation, and testing. It is important though to note that the audio programmer is part of the audio **team** and without the all the members working together nothing truly great can be produced.

In this report I will talk through all of the sections listed above and aim to discuss the role and importance of each.

## **1 / Specification**

Every solution starts with a problem, and therefore before one can effectively devise a solution the problem must be fully known. This is basically what the specification element of a project is. It is the analysing of a problem in order to better know a solution. In the perfect development system this is done in the pre-production stage [Hull, 2004], but in real life unless the development is trivial not all the hurdles are seen at the start and it becomes an ongoing process of iteration.

An example of problems that the audio programmer would have to find a solution for would be:

- Spatial positioning systems
- Real time audio filters
- Audio sample lookup database
- Reverb engines
- Sound content pipelines (appendix i)
- Audio file types
- Audio memory allocation

As it can be seen, there is the chance for a wide and varied amount of activities. Each one though can be dealt with in the same fundamental way, by being broken down into the four sections discussed here.

In the initial stages the problem is brought forward by a member of the audio team, in the form of a user requirements document. This document is generally a precise, informal statement of what is needed. The audio programmers then with the help of the team and this document will decide on the main goals of the solution, which will in turn give way to more problems. This is where iteration is invaluable, because the specification of one system will normally give rise to others [Wieggers, 2003]. After the main goals have been decided on, they will be sorted into low and high fidelity solutions, high fidelity solutions normally having more complex implementations. The problem will then be given a priority, with higher priority problems being dealt with a higher fidelity solution.

This initial stage will result in a focused goal for the solution, which can facilitate more in-depth analysing of the situation, a stage called 'requirements engineering'. The main objectives of this stage are to examine the problem in a more system centric way. In this I mean that thought is put into how the designed system will behave, testing that can be performed on it and tasks that will be needed to be completed [Sommerville, 1997].

These results will then be written up into a systems requirements document, which is for use by the audio programmers working on the system. It is much more detailed and technical than the users documents and generally contains:

- What you are trying to build
- How you will test what you have built
- The parts of the task that contain the most risk to other systems
- The parts of the task that are most likely to change
- How you will know whether or not you are successful

It is important that the requirements are separated from design, by being the elements that are **necessary** in the final product. The general purpose of the specifications stage is to make the proceeding stages more focused, and therefore easier and more productive.

## 2 / Design

The first stage of the design section is to choose the tools with which to create the system. Nearly always if the system is an in-game operation, it will be written in C++. This is because C++ is a highly adaptable, fast and powerful programming language [Ullman, 2006]. If the system is more of a 'tool' based routine to help sound and level designers, then C# or Java is an acceptable language because runtime speed and optimisation are not so much of a priority (appendix ii & iii).

Next it is very important (especially in more complex systems) to plan the architecture of the system. As this is key foundation of the design stage, and the pinnacle part of any program. The plan of the programs architecture contains the ways in which all parts of the program will talk and interact with each other. This includes the signal paths in more digital signal processing based solutions, and in what format the audio data will be in at each point in the system. It is important because the main aims of object orientated programming such as C++, is to abstract data from each other, and create generic program objects [Ullman, 2006]. This means that one system/object is unable to access the data of another system/object without precise code stating that it has permission to, a very important attribute of any complex computer game [Begault, 1994]. The architecture is normally planned using a graphic piece of software because it is easier to represent the complex connections visually, especially when communicating the design to a team of people (fig. 1).

I will now take a brief moment to explain what I mean by object orientated programming (OOP). It is a method by which each element or function within the program is given its own generic object definition, this definition can then be called every time that particular function is needed and also used to create a hierarchy of other objects with similar attributes (fig. 2). This lends itself to many different design paradigms and patterns, such as procedural abstraction, data abstraction and generic programming [Stroustrup, 2000].

Particularly good system architecture will make use of more than one of these paradigms. There is whole world of complexities behind programming paradigms which is beyond the scope of this paper, for readers who are interested I would highly recommend *C++ The Language* by Stroustrup.

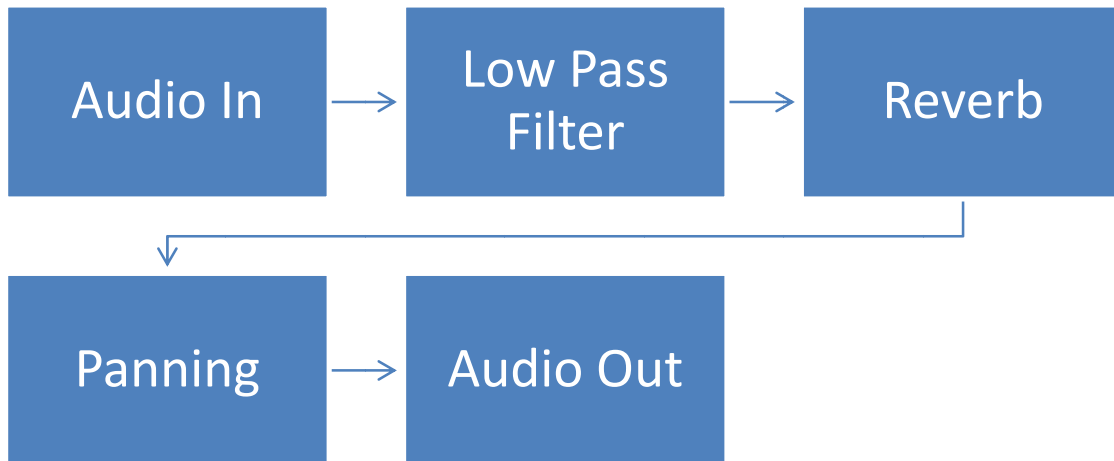


Fig. 1:- An example visual representation of a programs architecture.

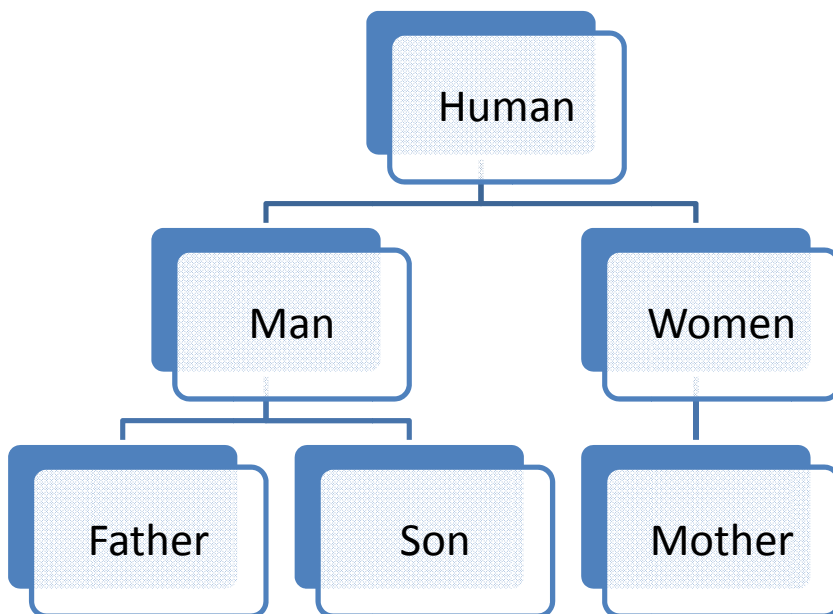


Fig. 2:- Graphical representation of hierarchical object oriented programming.

### **3 / Implementation**

Everything up to this stage has been very theoretical, based around ideologies of programming paradigms and system architecture. These sections are very important in gaining some space between the solution goal and the coding. It is all too easy to produce the required solution while in the process creating code that is unusable, due to being unmanageable, unchangeable, and badly integrated with the existing systems [Wieggers, 2003]. Therefore it is crucial (unless implicitly stated within the requirements) for individual code to be decided upon in this section, along with other such things as audio effects and digital signal processing [Begault, 1994]. It is within this section that the solution to the problem comes to life.

The first procedure that is carried out, is basic low fidelity prototyping of the relevant objects and systems within the overall solution. The main reason for this, is to create prototypes quickly, in order to iterate and find the most suitable and optimised solution to the defined problem. During this stage the audio programmer is not only looking for the most computer efficient prototype, but also the one that produces the highest audio quality. The trade-off to be debated is between audio quality, and the available operating system resources given to in game audio. It is important to note that the audio programmer is required to have objective audio listening experience (a 'good ear'), as well as programming experience, and it is this reason that there are not many audio programmers in the industry [Game Dev, 2007].

After the initial prototype, suitable solutions will arise which can be fitted into the designed architecture. These are then further prototyped with increasing quality but decreasing quantity, until only one high fidelity solution exists (fig. 3).

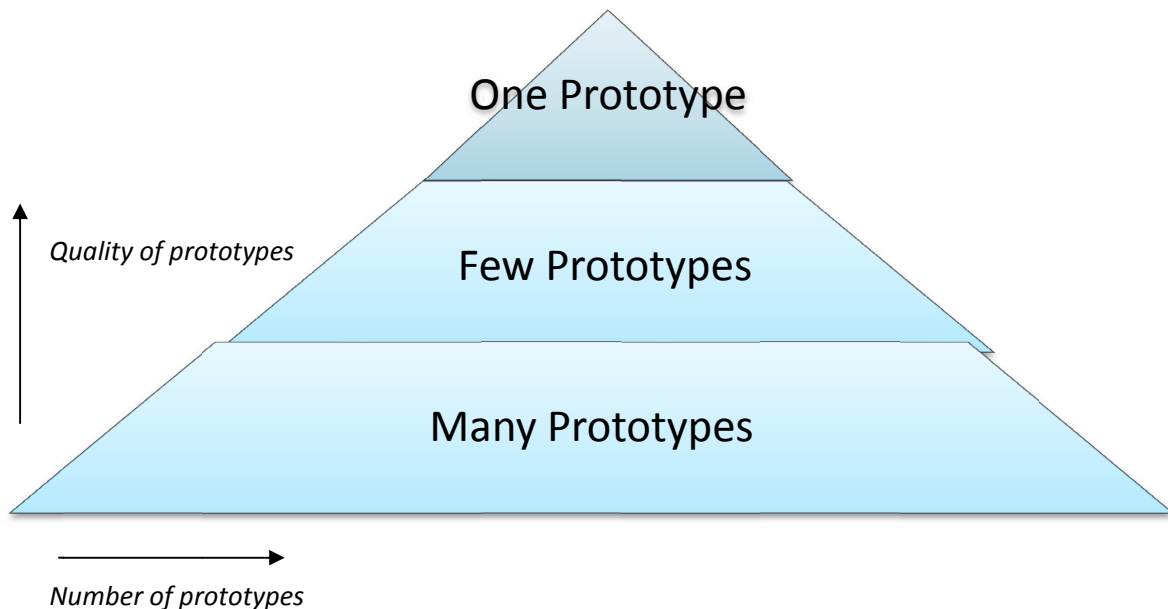


Fig. 3:- Pyramid method of system prototyping.

In this stage the various prototyped sections are pieced together to create the whole working solution. This is then tested against a copy of the full game code stored on the individual programmer's hard drive, a system known as source branching [Game Dev, 2007]. The testing is based around the testing requirements stated in the systems requirements document. It is devised to test basic to advanced user actions, as well as predicted weaknesses within the system. Further to this automated test 'bots' are available, which will enter the level and perform a predefined task respectively, for example pick up a box in every room. This is very useful when testing sound effect audio and ambiences. Now the most suitable prototype for the solution and integration into the overall system has been found, the next stage is to finalise the system and the produced code.

The final stage of integration into the main code base ('main line') requires communication with all members of the computer game development team, as it is critical that it is understood how their systems interact with the whole structure, and therefore with the newly integrated system. Something as small as using a 1Kb piece of memory that another system needed could crash the whole game [Game Sutra, 2007]!

## **4 / Testing**

While iterative testing is a fundamental part of the prototyping mentioned above, it is not the subject of this section. In this section I am more interested in the testing of the whole game after the designed, or solution system is integrated. The overall architecture of the whole game is often so complex, that one part can cause a detrimental and unpredicted effect on another part that it is not directly connected to. For this reason it is very important that all code and content that is added to the game whole is fully tested before release. Actually Electronics Arts for example hires a few hundred people just for the job of testers [Wiki, 2007]. The found errors or 'bugs' are then returned to the audio programmer in a database format, which indicates what procedures were being performed when the bug was detected. The programmer will then debug through the code to check if the error was in his/hers code or caused by another system's interaction. In general the last three to five months (depending on game complexity) are sent fixing or hiding real time in game bugs [Wiki, 2007].

In tool based solutions the testing that takes place is slightly different, being generally based around the tools usability and pipeline optimisation. Therefore tool testing is performed in very small groups normally the programmer and the design team whom are to use it.

## **5 / Iteration**

The sections mentioned above are not just performed through once to create a fully working finished system, quite the opposite they can be cycled through many times at various points during the production and development of a computer game. The most to common sections to oscillate between are the implementation and the testing, as the iterative prototyping method explained above lends its self to this. It is this act of iteration that means that the audio systems in place within the game are kept up to date with the needs of the game and the sound designers working on it [Begault, 1994]. Also when new systems and programs are created for use in other areas of the game, it is important that all the parts interact together in an optimised and high functioning way, which is up keeping with the overall architecture of the whole structure.

## **6 / Conclusion**

As the above text shows the audio programmer is an implicit member of the audio team, without whom the sound designers ideas and created audio would ever make it near the completed game. There expertises are needed in order to create the systems and pipelines required to get audio content into the game, and to manipulate the audio within it. It is these systems in combination with the audio content that make the in game audio interesting, non-linear and most importantly believable.

## / Appendix

### i / – Pipelines

In development terminology a pipeline is defined as a process or string of processes which convert, transmit, or transport game content into the game. For example from sound designers Pro Tools, to wave files, converted to game file type, and then put in memory by audio programmer.

### ii / - In Game Systems, and Tool Based Systems

In game systems are defined as a piece/s of code that performs a needed in game function (such as real time reverb), whilst tool based systems are programs that are used for getting content into the game such as level placement tools, and game file type converts.

## / References

Hull, E. *et al: Requirements Engineering*. Springer Press (2004)

Wieggers ,K: *Software Requirements*. Microsoft Press (2003)

Sommerville, I. *et al: Requirements Engineering*. Wiley Press (1997)

Ullman, L. *et al: C++ Programming*. Peachpit Press (2006)

Stroustrup, B: *The C++ Programming Language*. Addison Wesley (2000)

Begault, D: *3-D Sound for Virtual Reality and Multimedia*. Morgan Kaufmann (1994)

Electronic Arts University – In House Training (2006)

<http://www.gamedev.net/> Game Dev, accessed 2007

<http://www.gamasutra.com/> Game Sutra, accessed 2007

<http://www.codeproject.com/> Code Project, accessed 2007

<http://uk.playstation.com/> Sony Play Station 2 Homepage, accessed 2007

<http://www.xbox.com> Microsoft Xbox Homepage, accessed 2007

[http://en.wikipedia.org/wiki/Game\\_tester](http://en.wikipedia.org/wiki/Game_tester) Wiki on Game Testing, accessed 2007