Usage of Pd in Spore and Darkspore

Kent Jolly
Maxis/Electronic Arts
6121 N.Hollis
Emeryville, USA, 94608
KJolly@Maxis.Com

Abstract

The main focus of this paper is to describe the technical implementation of Pd into our game environment. The presentation will also cover some of the further usage of the tool in game with example patches and discuss some of the challenges involved. The work done became what is called EApd and represents our attempt to cast the tool in a production environment light.

Keywords

EApd, Spore, Electronic Arts, Games

1 Introduction

Interest in using a language like Max/MSP or Pd in games has been around for quite a long time, but an actual opportunity to use one of these languages is rare. EA had its own internal software that addressed many of the same ideas, but around 2002 that software was no longer in development, and left many teams in a position of having to come up with their own solutions due to concerns over support. There were two teams at EA interested in using Pd or Max at the time, Spore and Dead Space (The Dead Space team eventually gave up on either and instead used Lua, mainly for performance reasons described below). In the end it was the open source code and the clear division between the graphic UI and event/audio engine that made the decision in favor of Pd as opposed to Max/MSP.

Having decided to use Pd, the next area of work was focused on exactly how the software would be used. An important decision here was that we would not be using Pd as a dsp engine, but only for its event logic. This was done for several reasons. Audio engines were already there, and optimized for use in game. The tool was seen as something for sound designers to use as opposed to sound engineers, so it was assumed that writing dsp algorithms was not within the scope of application. The work that would need to be done to recreate a game audio engine was sizable and it seemed likely to reduce overall performance in the service of flexibility.

So, the first order of business was to remove audio engine elements from Pd and replace it with a set of objects that interfaced with our audio system instead. There was also interest in re-creating some objects from Max/MSP, most significantly the Coll object and the function table.

At this point I should also point out that EApd was not the only sound "player" in use. In Spore, EApd was something to be used in special case circumstances as opposed to a "plays everything" system. The reasons for having a system separate from EApd to play sound are of interest. There are two that are clear to me:

First, from a production point of view, it can be quite cumbersome to create patches to do something as mundane as just "play a sound", and so there needs to be some type of automation done in order to generate patches to play this type of sound, or instead, a larger patch created to deal with playing many different sounds in different contexts. In short, it quickly becomes an engineering task as opposed to a sound design task. In games, most sound designers are not nearly as good at designing event systems as they are sounds!

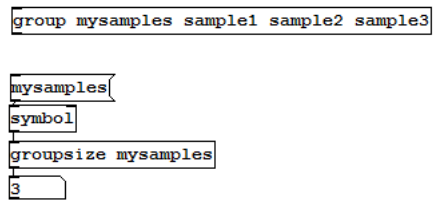
Second, performance of Pd was not as fast as desired. Often our sounds needed to spawn many instances, all with varying 3d positions and so on. Memory allocation on load time (due to text format of patches) was a problem, and large call stacks on crashes were unusual and cause for concern. I think anyone interested in getting Pd to be used in games needs to consider these aspects first and foremost.

Software like Pd or Max/MSP has a great deal of potential for use in games, but it is also not really well suited to game production environments. I hope that by describing the implementation, future attempts may gain from it. I should note that none of the software engineers who worked

on this project are still at EA, though they have been kind enough to answer a few questions from me in regards to the project.

2 The EApd audio object interface

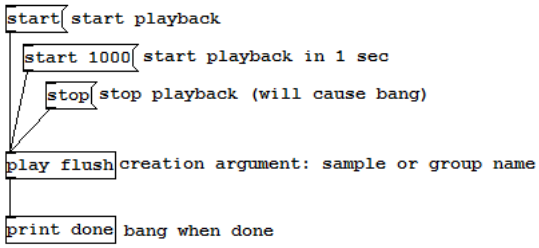
2.1 The Group and GroupSize objects



The group object is initialized with a group name, and a list of the member sample names. The definition of the group object is valid for the entire patch and sub-patches. Play objects can then refer to group object names for playback. The groupsize object will report the number of samples found in the group by symbol reference.

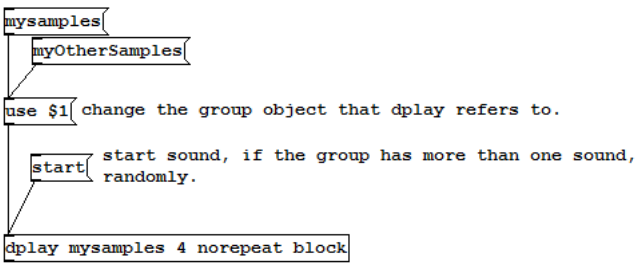
2.2 The Play and Dplay objects

2.2.1 The basics



The play (or dplay) object is the basic mechanism for audio playback in EApd. It takes either a sample name or a group name as a creation argument. A bang is output when sample playback is complete.

2.2.2 The arguments



The above figure shows the dplay object (dynamic play), dplay functions the same way as play however it lets you specify a new group or sample to play dynamically. The arguments shown here are as follows. Group (or sample) name, polyphony, sample pick method, polyphony handling.

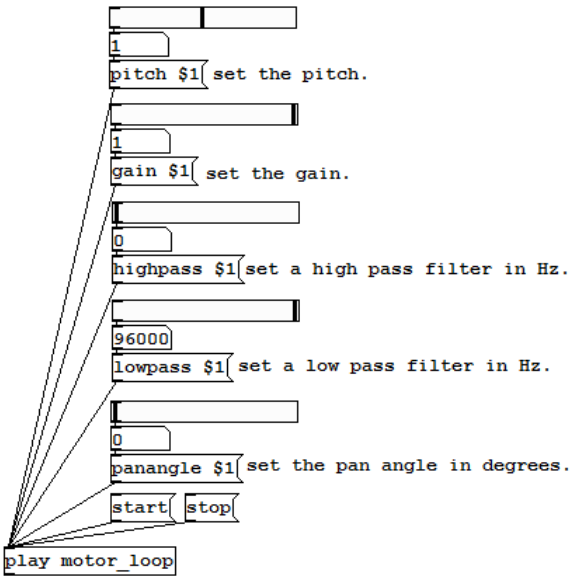
Group (or sample) name is self explanatory; the play object refers to the sample or group of samples named. Sending a use message can override the default.

Polyphony limits the number of concurrent instances the object can play (4 in this case).

Sample pick method can be either random, norepeat, or inorder.

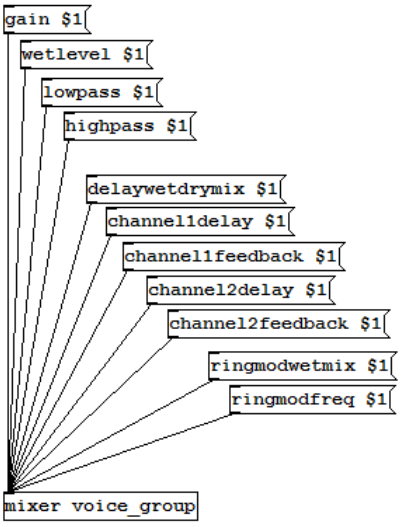
Polyphony handling arguments can be either steal or block. Steal will stop the oldest sound instance when the limit is reached, block will disallow start requests over the limit.

2.2.3 The parameters



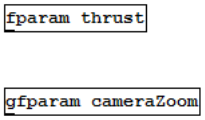
The parameters of the play/dplay objects; this is the basic set available. Not shown here are the x/y/z positional parameters used for 3d sounds.

2.3 The Mixer object



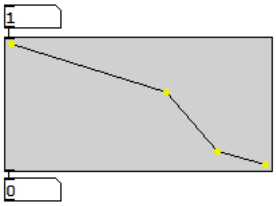
The mixer object is an object that can send messages to audio channel/bus effects as well as groups of sounds defined in the spore audio system (as opposed to groups defined in EApd itself). This is used to do mix automation of volume levels and effects based on game input.

2.4 The Fparam and GFparam objects



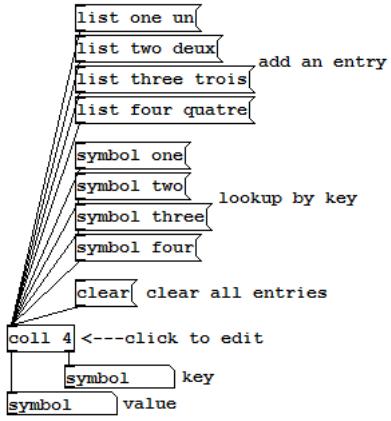
The Fparam and Gfparam (float parameter and Global float parameter) objects are the used to hook to game events. Typically a sound designer will request an engineer to hook up some state or parameter to Fparam or Gfparam once work has begun on a patch. The Fparam is local to each instance of a loaded patch, while Gfparams are global inputs and will be the same for any loaded patch.

2.5 The Function object



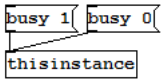
A Max/MSP style function object for editing response curves.

2.6 The Coll object



This is another Max/MSP like object, made largely because there was no easy list management that came with Pd and we were accustomed to Max/MSP methods.

2.7 The Thisinstance object



The Thisinstance object is used as a way to communicate back to the game system whether or not an instance of a patch should be destroyed. The case where this is often needed is during a fadeout process, where the game has finished playing the sound, but the sound and patch need time to fade out.

3 Exporting patches

Once a patch is complete and ready to be used in game, in EApd the sound designer must export this patch into a single file called a .pdr file. This is a process where all the sub patches are searched and included into one large patch file. Dependencies are removed and file management is simpler.

4 Functionality overlaps with Spore audio system

As stated earlier, EApd was not the only way to play sounds in Spore. Any sound in spore included with it a text property file that would include things like group membership, volume settings, and many other properties. These properties often overlapped with property settings that were set in an EApd patch, often leading to confusion as to which of the two was actually doing the setting of the property. This was an issue that was never fully resolved. In general it was up to the sound designer to make sure that sounds used for EApd patches were clearly set up to be controlled from EApd as opposed to the property file itself.

5 Conclusion

On both Spore and Darkspore EApd was a very powerful tool, it allowed for a great deal of creative flexibility with low cost in terms of engineering time. The drawbacks were that it was quite easy to overstep performance bounds, and as with any language like this, it is also quite easy to build things that are very difficult to understand later. Some of this is simply enforcing a disciplined approach, but I would suggest that future attempts to integrate systems like this should make a large effort to include higher level data visualization, debugging capabilities and also look for a way around the text file format for the patches themselves. I also believe it would be bet-

ter to consider the tool as the solution for all audio playback, as opposed to special cases usage. This might mean building automated pipelines for repetitive type work, etc.

6 Acknowledgements

My Thanks to EA, Lucy Bradshaw, Don Veca, Cyril Saint Girons, Justin Graham, Rajesh Ajjenegade and Mike Cormier.