Features:

All of the functionality mentioned in the project description of this project is accessed through a GUI that utilised Pure Data sub-patches. This ensured direct and simple access to all the available functions was achieved.

The orange coloured features are used to demonstrate the ability to perform a crossfade between two provided sounds. After the two sounds provided are initialised properly, the length of the transitions between both patterns is adjustable through the use of a slider, along with the volume of the output, the sound you wish to hear, and finally the adjustment of the speed of each pattern in BPM.

The blue section of the GUI shows an alternative way to load in sounds that are in a wav format with Pure Data, and the green section of the GUI demonstrates how a crossfade was achieved.

The red section of the GUI demonstrates the most important feature, the ability to generate two music patterns with some parameters to customise the output. The customisable parameters include the ability to select a custom speed which allows the program to compute the transition via the use of an audio crossfade and speed adjustment to allow for a smooth transition. It also includes the ability to change the speed of the "background music" standalone. As an added *coolness* factor, the background music is based on microphone input from the user, allowing for truly user generated sound.

Because this GUI was built using Pure Data sub-patches, it features the ability to view these sub-patched through the use of the Pure Data objects. This was done to ensure the program as written in a modular way, with all inner mechanisms hidden from the user.

Research

A lot of research was done in order to understand how to create generative audio. When creating our "Discovery" music patch, the work was an edited version of a previous patch [1] which helped recognise how to create a version that was relevant to what was desired from the project. Considering that our project detailed making smooth transitions with generative music, it was relevant that we did research onto how generative music was made. Further research included a reading on Synthetic game Audio with Pure data [2] and video game audio prototyping with pure data [3], these were both relevant research papers for this project, as the main motivation behind the project was to create a seamless transition between two separate game states, a "background" track and a "discovery" track. There were also numerous helpful videos that were utilised when creating the code for this project. Hermandez's pure data playlist [4] along with Wex's pure data playlist [5] were both watched in order to gain a greater understanding of what was possible through Pure Data, and how to do certain things such as read .wav files and allow audio to be looped. Another thing that was learned from researching Hermandez's and Wex's videos was how to increase or decrease the speed of each pattern in BPM. This was done by finding the sample rates of each sound, and then creating a slider that would be used to adjust this sample rate, which was used to make a sound go as slow as half its original speed, and as fast as double its original speed. The official Pure Data forums were also used to perform research, as when the project came to a halt due to a lack of understanding about Pure Data, help forum posts solved many issues [6]. The research for the project also consisted of finding sounds to use that were not generative, this was decided to ensure that when we would use tools such as crossfading and transition speed changes, it would be known they are working correctly. For this project FreeSound.org was used, and the two sounds that were chosen were a guitar by a user known as timcam [7] and a piano by a user known as Lemoncreme [8]. These sounds were chosen based on the research of Dr Kelly Fitz [9]. Kelly's website lists a number of examples of sound morphing, and the decision was made to use two sounds that are pleasant to listen to when morphing, but that could still easily be distinguished from one another so that a user would be aware that a sound morph had taken place. Although sound morphing was not implemented in the end, the teachings of this research were useful when creating the crossfader for the project. Other than outside research, it is also important to mention research done on previous lab works, for example lab 4 [10] was used in order to allow the project to incorporate packing and unpacking, which was used in order to create custom transition speeds through the use of a slider.

Code:

The code for the orange part of the GUI firstly works by having two initialisation bangs. Once the user bangs one of them, a file select window will open, in which the user should open one of the two provided .wav files based on their choice. This information is then resized, and read into two separate arrays for the left and right channel with the "read -resize $1 channelL channelR" object. In order for our sound to loop, the arrays would be called instead of having the read the sound in from a file every time. The BPM was found by using the "expr 44100 / $f1" method, and a slider was created that would go from half of this value to double this value, so that the speed of each pattern in BPM was adjustable. The transition between sounds was created using an attack sustain release, which packed values from a slider so that the speed of a transition was customisable. The speed of both sounds was made customisable through the use of the equation of a line ($y = mx + c$) so that the speed of both sounds could be adjusted at the same time. Through the use of a metro, a slider was created that displayed the audio over time, and allow for the audio to loop.

The code for the blue part of the GUI works by having the sounds opened through Pure Data rather than by a user through a file select window. The sound loops by sending a bang after every read statement back to the open method, so the .wav is opened every time the sound begins, which is more strenuous than the version in the orange part. A slider acts as a makeshift crossfade that allows a user to slide between the two sounds opened.

The code for the green part of the GUI works by utilising a crossfader sub-patch. This sub-patch has three inlets, and these inlets are used to open two sounds, and to have a slider that will allow a user to crossfade between these two sounds. The speed and volume of the sounds are crossfaded to ensure a clean transition between the two sounds. The way in which this crossfader object works is by taking in two different sounds (be they generative or pre-loaded) and gives an output to a digital – audio converter based on where the slider is positioned. This patch was also used in a later part of the code, in which a crossfader generative patch was made to exclusively perform crossfades between generative audio.

The code for the red part of the GUI works by utilising the crossfader from the previous section, along with two separate methods for generating music. Firstly a line object is used in order to allow users to select different speeds at which they gradually want to move from one sound to the other. Background mic generative is the first piece of generative music, and is created based on microphone input from the user. It makes use of the external freeverb object that was included in the ZIP file along with the Pure Data submission. The other piece of generative music does not take microphone input, and instead works by utilising the work of Martin Brinkmann. A member of our group used this base code, and added several sub patched to it in order to allow the code to generate a style of music that we found suitable.

Conclusion:

The idea for this project was spurred from a member of our group creating generative music in their own time. It was then mentioned to us that this was something sought after in the modern sound world, particularly in the video game industry where repetitive music can diminish the enjoy ability of a game. This was the reason why we as a team undertook this project, as working in the gaming industry appealed to a majority of us.

Upon reflection, the project was unable to perform its suggested extra feature, which was to compute morphing between the two patterns in the frequency domain. This would have ensured that the transition sounded even better then it currently does. This project also did not utilize masking, and even though masking was a lazy solution to this problem, it would have been a good idea to demonstrate that it was something we could accomplish, and perhaps create a version that utilised both a cross fade and masking, so the two could be compared as to which sounded better.

This project was a pleasant experience, and there isn't much room for improvement, perhaps students taking this course in the future could have the successful ideas from this year provided to them as possible projects (considering we are the first year of the revised course, it is obvious why we did not have this). The individual report deadline, although fair, could have also been slightly longer after the due date of the code, but this is a minor issue.

Bibliography

[1] Martin Brinkmann, 2011

[2] Synthetic game audio with Puredata (Summary),
Andy James Farnell
andy@obiwannabe.co.uk

[3] Video Game Audio Prototyping with Pure Data,
Leonard J. Paul
Vancouver Film School, 1380 Burrard Street, Vancouver, BC, V6Z 2H3, Canada
info@lotusaudio.com

[4] https://www.youtube.com/playlist?list=PL12DC9A161D8DC5DC
cheetomoskeeto / Dr Rafael Hermandez
last updated on 2 Jul 2014

[5]
https://www.youtube.com/playlist?list=PL0dqIhYnzlnPeQAC5mRzKq5HLwfBGRYaO
ob wex
last updated on 27 Nov 2015

[6] http://forum.pdpatchrepo.info/

[7] https://www.freesound.org/people/timcam/sounds/97775/
timcam
May 28th, 2010

[8] https://www.freesound.org/people/Lemoncreme/sounds/186942/
Lemoncreme
April 30th, 2013

[9] http://www.cerlsoundgroup.org/Kelly/soundmorphing.html
Sound morphing and modelling
Personal research web page for Kelly Fitz, Ph.D.

[10] https://moodle.royalholloway.ac.uk/course/view.php?id=1459
Lab 4
Nuno Barreiro