Study on effectiveness of procedural, interactive and adaptive sounds in videogames

Alessio Tosto
Dept. of Computer Science
University of Milan
Milan, Italy
alessio.tosto@studenti.unimi.it

***Abstract*—In the field of videogames, sounds are a key component for immersion and to provide audio cues that players can use to adjust or improve the way they play the game. This work proposes a study that aims at verifying how effective interactive and adaptive sounds are at improving the player performance of a videogame. The study is conducted by the use of a game prototype that features audio cues procedurally generated in real-time that correctly match the interactions player can achieve while playing and by analysing the final performance of a given amount of test subjects in different circumstances: without audio cues, with audio cues with high latency and with audio cues with low latency, in order to verify if interactive and adaptive sounds do effectively improve the player final results.**

I. Introduction

A. Problem Overview

Videogames are a work of art that feature components coming from a broad range of fields, including the audio one. Audio in videogames is commonly divided into two macro types: background music/ambience and sounds/effects with the second category furtherly being separated in UI sounds and interactive sounds.

Interactive sounds are particularly important in a simulated experience since they are the main source for immersion, providing the player a way to reinforce the sense of being inside a living world. Interactive sounds are also a key element in providing audio cues that a player can use to adjust the way he approaches the game to improve his own performances. As a confirmation for this thesis, audio cues provided by interactive sounds are a staple element for blindfolded speed runners, a category of players that complete videogames in the fastest way possible while being completely unable to view the display and so relying on audio cues to be able to localize where they are and what the situation around them is in the simulated experience.

This study focuses on this kind of sounds and wants to analyse how much they affect the final performance of the player and, at the same time, wants to determine if low latency is a requirement for the effectiveness of adaptive sounds in videogames performances.

B. Method Overview

To conduct this study, a game prototype has been built, using the Unity game engine, that features a controls scheme entirely based on accelerometers, providing a more immersive experience for the players. For the audio component, interactive and adaptive sounds are generated procedurally and played by an external instance of Pure Data running a patch and communicating with the main game executable thanks to messages shared via OSC (Open Sound Control) protocol. The prototype is composed by five distinct levels with an average approximate expected completion time of the five levels being around 1 and 2 minutes. During the gameplay, two statistics will be collected: the number of times the player will fail a level and will be forced to restart from the start of it, and the overall time each single level took to be completed.

The experiment is conducted on a set of subjects of different age spacing between 22 and 50 years old. Every subject will first be allowed a free play of the prototype to get accustomed with the controls and with the levels included in the game.

After the free play, the subjects will play again the prototype for three more times with the setup being slightly different in each of the runs. The first run will be with interactive sounds enabled with low latency (80ms), the second run will be with interactive sounds enabled with high latency (250ms) and the final run will be with audio sounds disabled.

The results coming from all the runs from all the candidate subjects will then be compared to raise conclusions about this study in relation to the data collected.

C. Related Work

There are several works that address the importance of interactive sounds in videogames. The study conducted by Jonathan B. Fish named “Interactive and adaptive audio for home video game consoles” [2] assesses, for instance, the importance in home video games to include interactive and adaptive audio to convey a sense of interactive partnership between the player and the game itself. The study conducts analysis from data collected from multiple sources, including existing videogames in the industry, and raises the possibility of interactive and adaptive audio to work, not only as a medium for immersivity, but also to empower the player. This claim is also reinforced in a similar study conducted by James Boen [3] where it comes out how sounds offer three important features to a virtual experience: environmental, impact and diegetic. The diegetic component refers at how sounds can be used by players as a vehicle of information, thus capable of guiding the player towards their objectives and thus, being a key gameplay component that can help plasm the actual way in which players approach the game itself.

D. Paper Organization

The reminder of the paper is organized as follows. Section II features a more in-depth view of the game prototype used during the experiment. Section III shows the results of the playtest sessions and the evaluation of the outcomes. Section IV provides a brief recap of the motivations that lead to the experiment and how this study is effective at confirming or dismissing the claims coming from literature. Further ideas on how to improve any future study are also provided in section IV.

II. Prototype Implementation

A. Used Tools

The game prototype developed for this study is realized using Unity v.2022.3.20f1 game engine. The audio component of the prototype is entirely handled in Pure Data (PD), an opensource visual programming language with primary focus on multimedia developed by Miller Puckette [4]. Sound Design Toolkit (SDT), a collection of physically informed sound synthesis models with the goal to enable research in Sonic Interaction Design [5], in form of a framework, is used with PD to procedurally synthesize sounds based on solid interactions. The original idea was to develop a game prototype that would be later exported for Mobile (Android and iOS) through the usage of LibPdIntegration, a wrapper for libpd to incorporate Pure Data inside Unity natively [6]. This approach would have allowed an easier setup of the prototype, due to it not relying on any external tool, like Pure Data GUI, that could result relatively complex to install and properly configure for end users thus allowing for a broader range of subjects for the experiment. Unfortunately, this wrapper does not support Android target and lacks support for external libraries (only vanilla PD is supported), making SDT impossible to be used. After some attempts to eventually circumvent these limitations, efforts shifted into producing a prototype with Windows as target and relying on Open Sound Control (OSC), a messaging protocol particularly well suited for communication between multimedia systems [7], for the communication between Pure Data GUI and the exported Unity executable. This made the setup of the prototype relatively more complex than expected and, to deal with this change, the physical setup for the experiment had been adjusted to a fixed desktop PC with everything properly configured so that end users did not have to manually setup the prototype on their own machines. This led to a smaller number of subjects being used for the experiment.

B. Prototype Structure

The prototype is structured into six different scenes: an introduction scene and five game scenes, each one covering one of the five levels included in the game.

The introduction scene serves as an explanation of what will happen once the game prototype has been started. As such, it serves to inform players on what the experiment consists of.

Each playable level is structured so that player must roll a ball using accelerometer from the starting point down to the goal area (marked with a semi-transparent cyan effect). Once inside the goal area, player must not exit the goal area for three seconds in order to complete the level.

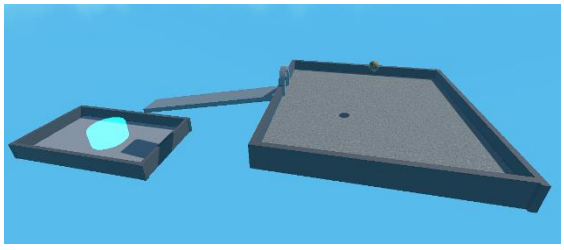

Fig. 1: First playable level of the prototype.

After completing the level, the player will be shown the completion time and, after completing the final level, he will be shown a recap on the completion time of the single levels and a total playtime combining the five playable levels completion times.

C. Controls

To provide a sense of immersion for the player, controls had been implemented through the usage of accelerometer. In the original plan for the prototype (Mobile target), accelerometer support would've been natively implemented in Unity but, for the final prototype (Windows target), accelerometer support had to be implemented manually since Unity does not provide support for accelerometer for any modern controller on Windows target.

To achieve this, a DualSense controller has been selected as the one to be used during the playtests and manual analysis of the reported values from the human interface device (HID) protocol of the controller connected via USB cable to the target PC had been performed thanks to the Input Debugger feature in Unity editor. From this analysis, combined to the HID descriptor struct available on ds5vita opensource project [1], it had been possible to add accelerometer detection into Unity DualSense HID layout.

Display Name	Layout	Type	Form	Offse	Bit	Size	Optio	Value
DualSense HID	DualSense	DualSense	DSv	0	0	224	-	System.Byte
accl X 0	Button	ButtonCon	BYT	22	0	8	-	0,9411765
accl X 1	Button	ButtonCon	BYT	23	0	8	-	1
accl Y 0	Button	ButtonCon	BYT	24	0	8	-	0,4862745
accl Y 1	Button	ButtonCon	BYT	25	0	8	-	0,1176471
accl Z 0	Button	ButtonCon	BYT	26	0	8	-	0,3058824
accl Z 1	Button	ButtonCon	BYT	27	0	8	-	0,01960784

Fig. 2: Accelerometer implementation in DualSense HID layout.

With the layout properly allowing to access accelerometer values inside C# scripts, the only major roadblock left for using accelerometer data feed in the prototype was given by the fact Unity allows only single bytes length variables to be used for HID data but accelerometer data, as shown from ds5vita struct, is 2 bytes length (unsigned short). To deal with this problem, we expose two different variables for each accelerometer axis (X 0, X 1, Y0, Y1, Z0, Z1) and, when we want to use the actual final value in the game scripts, we combine the given values through bitwise operations carried over with System.BitConverter class.

```
private static int processAccelData(float
accel0, float accel1)
{
    byte[] accelRaw = { 0, 0 };
    accelRaw[0] = (byte)(int)(accel0 *
0xFF);
    accelRaw[1] = (byte)(int)(accel1 *
0xFF);
    int accel =
System.BitConverter.ToInt16(accelRaw, 0);
    return accel;
}
```

Fig. 3: Function used to get proper accelerometer data inside scripts.

D. Audio

The interactive and adaptive sounds featured in this prototype are substantially two: the rolling sound coming from the ball movement and the impact sound when the ball comes in

contact with walls or new surfaces. Both these effects are managed with a single macro patch running on an external instance of Pure Data. This patch features two samples from the SDT framework (rolling and impact samples) opportunely adjusted and fine tuned in their objects settings to properly simulate a rolling ball, in the first case, and the sound produced by a ball impacting with surfaces with low roughness, for the second case. Originally, the idea was to also feature a third element in the patch to handle bouncing effect separately from the impact ones, although, the bouncing sample featured in SDT is dependant on bouncing timings which are handled, in our case, by Unity physics engine thus making it not suitable for this instance. The impact sound so is used also for bouncing sounds since, generically, bouncing is nothing more than an impact with the surface and correctly simulated by the impact sample. On top of these two samples, code to handle intercommunication between Unity and Pure Data has been added, making use of OSC. OSC is used as a unilateral channel, serving Unity to send updates and requests to Pure Data. More specifically, OSC is used in Unity to send two messages to Pure Data: the ball speed (as a /speed message) which is sent once per frame by deriving the value from the physics simulation, and to inform Pure Data whenever the ball comes in contact with a new surface (thus requesting an impact sound playback).

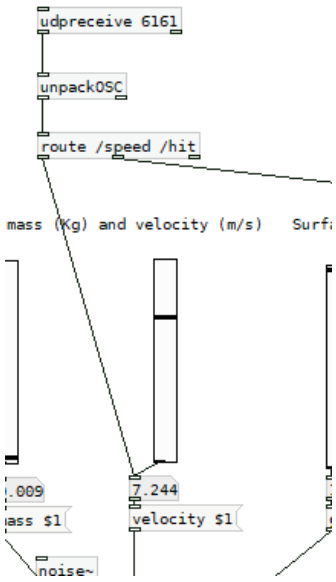

Fig. 4: 'speed' message handling in Pure Data.

The speed value is sent to the rolling sample to replace the velocity parameter. This effectively adjusts the final rolling sound to be representing an object rolling more or less fast. One special case, although, is handled manually, which is when the ball is moving mid-air because of a fall or a jump. In this instance, rather than sending the ball speed derived from the physics simulation, a hardcoded value of 0.0 is sent to Pure Data every frame until the ball impacts again with a surface thus making the rolling sample produce a completely silent sound.

For the impact sound instead, an /hit message is sent from Unity which Pure Data interprets as a request to play a hardcoded impact sound with specific settings on the underlying objects used by SDT which simulate a very short impact sound where a spheric object impacts with a low roughness surface. This has been set like that so that the simulated impact could properly fit also bouncing sounds.

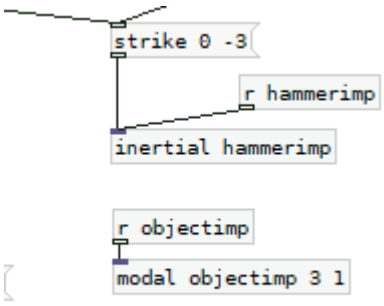

Fig. 5: 'hit' message handling in Pure Data.

III. Evaluation

The game prototype has been put in test by letting players of different ages play it four times in sequence: a free play to get used with the controls and the general feeling of the game, and then three testing playthroughs with different settings for the audio setup: the first time they played the prototype with audio enabled and low latency (80ms), the second time they played with audio enabled and high latency (250ms) and, the last time, they played the prototype with no audio at all. Two metrics had been collected during the testing playthroughs: the completion times for the levels and for the whole playthrough, and the number of times players did fall out of the level and had to restart from the start of the level. Beware that when a fall occurred, time for the running level was not reset, thus accumulating time across retries.

A. *Low latency results*

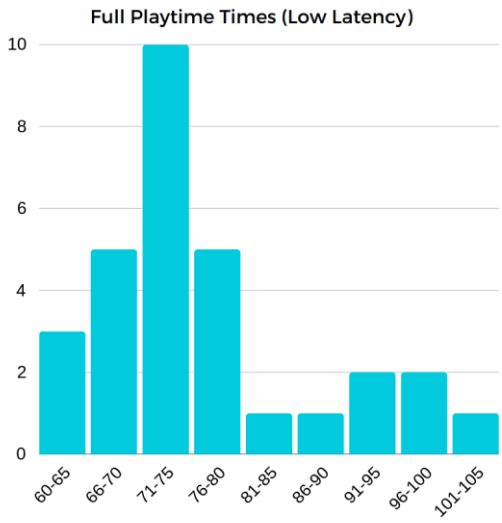

Fig. 5: Completion times for low latency tests.

Playtests with low latency audio shown, as expected, the best results around the three typologies of tests ran. As shown in the graph above, the majority of players (77%) completed the prototype in less than 80 seconds, with the biggest split (33%) being with a completion time between 71 and 75 seconds.

The best times recorded had been the following:

- Level 1: 00:10
- Level 2: 00:12
- Level 3: 00:13
- Level 4: 00:14
- Level 5: 00:10
- Total: 01:01
- Sum of best splits: 01:00

The worst times recorded, instead, had been the following:

- Level 1: 00:15
- Level 2: 00:14
- Level 3: 00:24
- Level 4: 00:21
- Level 5: 00:41
- Total: 01:41

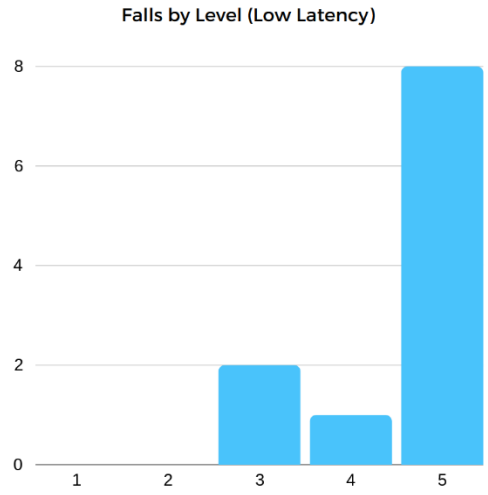

Fig. 6: Falls chart for low latency tests.

During the playthroughs, there had been a grand total of 11 falls, with the majority of them being concentrated in the last level (73%) and with no falls at all during the first two levels. This is particularly interesting because each level had been designed in a way to challenge the players in specific components of the game: the first level is a simple one to get used with the environment, the second level challenges the ability to rapidly change the direction to follow, the third level challenges weight balancing, the forth level challenges the ability to maintain sustained speed and the last level challenges the ability to accurately limit your speed and precision during free falls, which is arguably the hardest skill to master across the ones tested.

B. High latency results

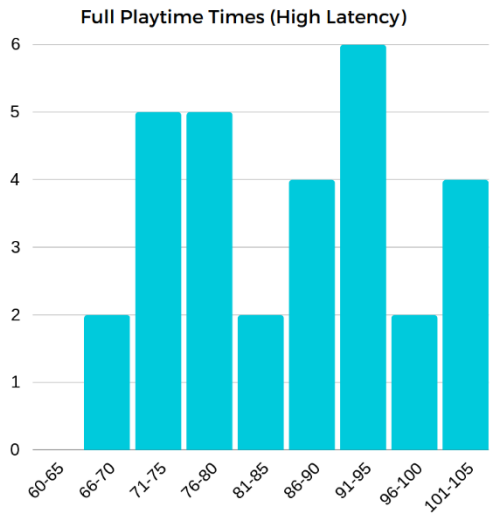

Fig. 7: Completion times for high latency tests.

High latency playthroughs showcased a radical shift in the results. As shown in the graph above, there is no more a distinct splice with a predominance in occurrences, but the results are more variegated. 53% of the players completed the prototype in more than 85 seconds with the split shared across most players being the one between 91 and 95 seconds (20%). An interesting note is that no player managed to complete the game in less than 66 seconds. These results align with the thesis of the importance of low latency audio for gameplay coordination.

The best times recorded had been the following:

- Level 1: 00:14
- Level 2: 00:12
- Level 3: 00:17
- Level 4: 00:14
- Level 5: 00:18
- Total: 01:18
- Sum of best splits: 01:16

The worst times recorded, instead, had been the following:

- Level 1: 00:21
- Level 2: 00:21
- Level 3: 00:31
- Level 4: 00:24
- Level 5: 00:49
- Total: 01:45

These results sign a significant worsening compared to the low latency ones, with a predominance still on the last level but not so marked as in the low latency one, as a prove that all skills are affected by the difficulty in coordinating movements introduced by high latency.

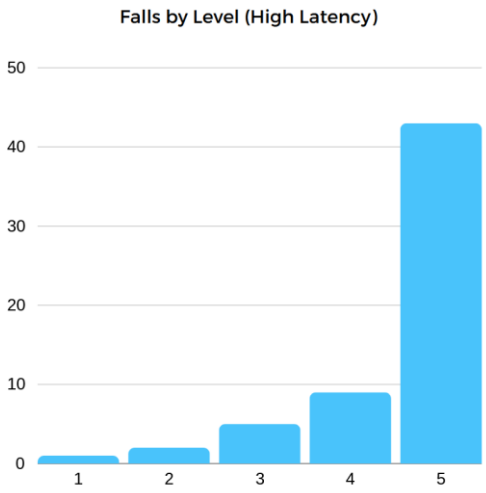

Fig. 8: Falls chart for high latency tests.

The falls chart shown above shows that falls happened also in levels that previously, with low latency, were not a problem for any player (level 1 and level 2). There has been also a massive increase on the number of falls, totalling to 59 with the majority (71%) still happening in the last level.

C. No audio results

Lastly, playthroughs with no audio at all had been put into test. These results are particularly interesting since allow us to check if high latency in audio is detrimental for the final experience or if it's preferable to still have audio but in desync with what is shown on the screen.

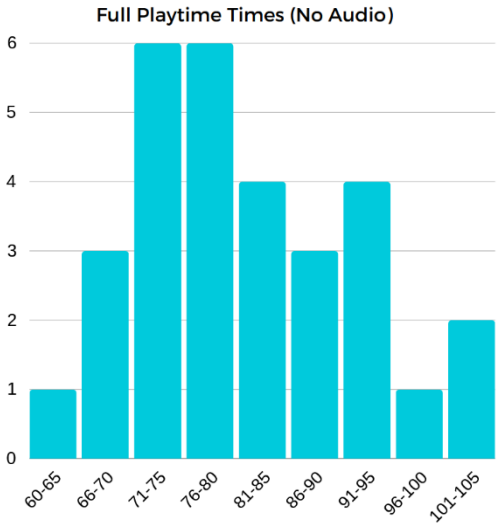

Fig. 9: Completion times for no audio tests.

The completion times, shown in the chart above, tell us that, whilst results are more widespread across all splits, similarly to high latency results, there is a major occurrence for lower times similarly to low latency results instead with 67% of the players managing to complete the prototype in less than 85 seconds. The splits with major occurrences had been the two splits with timings between 71 and 75 seconds, and the other one with timings between 76 and 80 seconds (20% each). There had also been a single player that managed to complete the prototype in less than 65 seconds, whilst the same achievement did not happen with high latency audio.

The best times recorded had been the following:

- Level 1: 00:10
- Level 2: 00:12
- Level 3: 00:13
- Level 4: 00:14
- Level 5: 00:10
- Total: 01:03
- Sum of best splits: 01:00

The worst times recorded, instead, had been the following:

- Level 1: 00:15
- Level 2: 00:20
- Level 3: 00:27
- Level 4: 00:24
- Level 5: 00:43
- Total: 01:42

These results are particularly significant because the best times for each level are the same ones as low latency runs and the worst full playthrough completion time is very similar to then one recorded with low latency audio. This proves the point that high latency is, actually, detrimental for the average player and that, preferably, no audio allows to achieve better results.

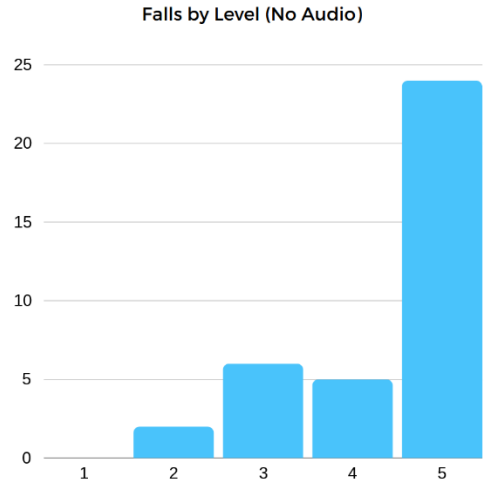

Fig. 10: Falls chart for no audio tests.

The falls chart shown above showcases, as with low and high latency results, an average difficulty into facing the last level of the prototype with the wide majority (65%) of the falls occurring on that particular level. There had been a total of 37 falls across all levels and no fall occurred during the first level, unlike what happened with high latency playthroughs.

IV. Conclusions

After considering the results shown in the previous chapter, we can conclude that interactive and adaptive sounds

do enhance the average performance of videogames players but only if those sounds are implemented correctly and with a low enough latency to not make audio cues be desynced compared to what happens on screen. This is proved by the completion times results for the three playthrough modes and by the difference in falls recorded between modes that highlighted how high latency in interactive and adaptive sounds is detrimental for the final performance of players. The detrimental effect of high latency in interactive sounds largely overcomes the benefits that low latency audio could bring to the player, as shown by the radical shift that there in the results between low latency and high latency audio, in comparison to the mild shift encountered between low latency and no audio tests; as such, if low latency audio can't be achieved due to particular restrictions in the software or hardware used, it's best to leave the game with no interactive sounds playback at all to not hurt the sense of immersiveness and ability of players to properly play the game.

While the tests performed can be considered exhaustive for the thesis, a more in-depth test could be run with a larger testers pool and a longer prototype so to assess more details and fine tuning the results even more.

REFERENCES

[1] NightlureSS, 2022, https://github.com/hedhehd/ds5vita/blob/main/main.c#L25-L98.

[2] Jonathan B. Fish, "Interactive and adaptive audio in home video game consoles", Simon Fraser University, 2003.

[3] James Boen, "Sound Video Games: How Sound Is an Important Aspect of the Virtual Experience", ART 108: Introduction to Game Studies, 2021.

[4] Miller Puckette, "Pure Data: another integrated computer music environment", 1970.

[5] Stefano Baldan, Stefano Delle Monache, Davide Rocchesso, "The Sound Design Toolkit", 2017.

[6] LibPdIntegration, 2023, https://github.com/LibPdIntegration/LibPdIntegration.

[7] Angelo Fraietta, "Open Sound Control: Constraints and Limitation", 2008.