Technical University of Denmark

02826 Sound Design for Digital Media

Final report

Egill Ingi Jacobsen s172759
Sigurbjorn Jonsson s172581

Date of submission: December 18th, 2017

Contents

1 Introduction

The project is a sound design for a computer game that was created in the course 02823 Introduction to Computer Game Prototyping. The game, When the Ninjas stole Christmas, is a 2D platformer about Santa's journey to save Christmas after a group of naughty ninjas stole it. Neither of us has ever spent any time in detailed sound design before so we are both excited for what lay's ahead. Our plan is to experiment with different kind of ideas to get the most out of the experience. Both of us have degree's in computer science but also has one of us a degree in electrical engineering and the other good experience in music so that brings different perspectives. The plan is to design sounds for Santa's movement, his contact with the environment, background noises and some soundtracks. We plan to utilize Pd as much as possible but also experiment with real-world noises as a substitute for the real thing and by doing so trick the person playing the game that he's hearing a natural sound.

As the Pd community is quite small and not a lot of help online we know it will be a challenge creating some of our custom sounds but with time and experimentation, we believe we will get better and come up with some smart designs.

2 Detailed Description and Features

2.1 Santa's activity

2.1.1 Move right and left

As the platforms that Santa walks on have snow on top of them we want to create a crunchy sound that comes when you walk on top of snow. If the player is running you should hear it more rapidly.

2.1.2 Jump

As there isn't some universal sound that comes when somebody jumps, we sought inspiration from Super Mario and want to create a bouncy type sound. The technical difficulty here is to put into words how it sounds.

2.1.3 Throw

The throw sound we envisioned to have in our game is something similar to when a racket is swung rapidly, but lower in frequency and with a volume modulation. In technical terms, we wanted a low-frequency sound that would rise a little bit in frequency with time. The volume of the sound should start by rising with time and then it should decay to zero.

2.1.4 Low in health

When the player's health gets low we want to play a sound for a short time to let the player know, in case he doesn't notice it immediately in the user interface. The sound effect we wanted to play for this is a deep-toned alarm sound.

2.1.5 Death

When the player loses his last health point he dies. In that event, we want to play a sound indicating his death. Originally we wanted to play a sound indicating failure, such as "whhaa whhaa whaaaaa", but we didn't manage to implement that to our satisfaction, therefore, we made another sound instead, that is a deep bass that gets deeper over time until it fades out indicating something bad happened.

2.1.6 Player Taking Damage

When the player is hit by an enemy attack or an environmental hazard, such as bombs or icicles, the player loses one health point. To indicate that the player is taking damage we want to make a sound effect. The sound effect we want to have is something similar to when we hurt our selves, i.e. a sound that goes "ahhh".

2.1.7 Going down chimneys

It's a popular belief that Santa uses magic when going down chimneys to deliver presents under the Christmas tree. In our game, when Santa goes down chimneys he is either completing a level or traveling to a sub-level. We decided that it would be fitting to play some kind of a magical sound when he does this. By a magical sound, we are referring to a sound that is a mix of high frequencies dancing around, creating a cheerful and "magical" effect.

2.1.8 Getting blown up in the air by chimneys

When a smoke blows out of a chimney in the game the player will be shot up into the air if he's on top of the chimney. We thought it would be important to make a sound for when the smoke is blowing out of the chimney. In real life, you would need a huge amount of pressure so we imagined that it would sound a lot like gas. How much the player hears the sound should depend on his distance from the chimney.

2.1.9 Eat and drink milk and cookies

In the game when the player picks up milk and cookies he regenerates one health point. When this happens we want to play a sound of him drinking, something like "glugg glugg glugg".

2.2 Environment

2.2.1 Gift

The decision for how picking up a gift should sound like was heavily influenced by what games of the platformer genre we had played. We immediately knew what a popular sound for picking up coins in games sounds like. In words, we would describe it as a "bling". A mix of sounds at higher frequencies which builds up steadily to a maximum before decaying again and dying out.

2.2.2 Bomb

The bomb that was created in the game has two elements. First, a fuse goes off when the player walks to close to it and after it burns out an explosion happens. For a fuse we watched videos of fireworks being lit, that was the perfect sound to implement. When we thought about what an explosion sounds like, we thought of a sharp attack at low-frequencies followed by a low frequency rumble.

2.2.3 Icicle

In the game when a player walks under an icicle it falls down to the ground. If it hits the player or just the ground there should be a sound played of it breaking. The sound should be a glass type sound that is played multiple times for a few milliseconds as the icicle shatters.

2.2.4 Weather

Level 1 takes place on the North pole so it should be windy and snowing.

2.3 Soundtrack

To keep the person playing the game more entertained and have the game more in the Christmas spirit we have arranged famous Christmas melodies to be played under different menus and levels throughout the game. The tunes we have select are:

1. Silent Night by Franz Xaver Gruber
2. Jingle Bells by James Lord Pierpont
3. Deck the halls by Thomas Oliphant

3 Implementation

3.1 Game Sound Effects

3.1.1 Player Walk and Run

To start with when making a sound for the snow crunching beneath the player's feet we tried making a Pd patch with different band-pass filtered noises. We got close to a sound that we thought to be usable but were never entirely satisfied. Because of that, we tried to change pace a little bit. Outside there was newly fallen snow, so we went out with our phones as recorders and recorded different sounds the snow made, f.ex. an actual crunch of the snow beneath our feet. When done we loaded our recordings into Audacity where we listened to our sounds and extracted the two parts that sounded best in our ears, one of them was a slow crunch and the other a faster crunch. We then loaded these parts of the recordings into Pd where we combined them at variable speeds, i.e. we play the recordings randomly at 0.5-1.5 times the original speed. To add a little bit more richness to the sound we also added a bit of low-frequency noise. Figure 1 shows the Pd patch that implements this algorithm. The pd patch can be found at "/Sounds/Pd patches/Santa's activity/walk_run.pd".

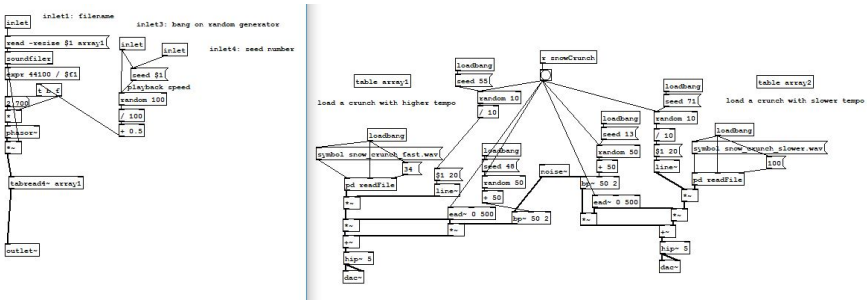

Figure 1: A Pd patch that combines two audio recordings of crunching snow. Each recording is played at 0.5-1.5 the original speed.

3.1.2 Player jump

To make a Super Mario type jump sound we understood that we had to take some waveform and continually raise its frequency with time[6]. Instead of using a simple oscillator with a sine wave we decided to experiment with other types of waveforms. We generated a square waveform with a sinesum message, and then used a tabosc4 block to look up values in the waveform. The sinesum is essentially an easy way to implement additive synthesis. We used a vline envelope to sweep the frequency of the waveform. All this put together gave us the sound effect we were looking for after a little play with frequencies. Figure 2 shows the Pd patch that generates a sound with continually rising frequency for short period of time. The pd patch can be found at "/Sounds/Pd patches/Santa's activity/jump.pd"

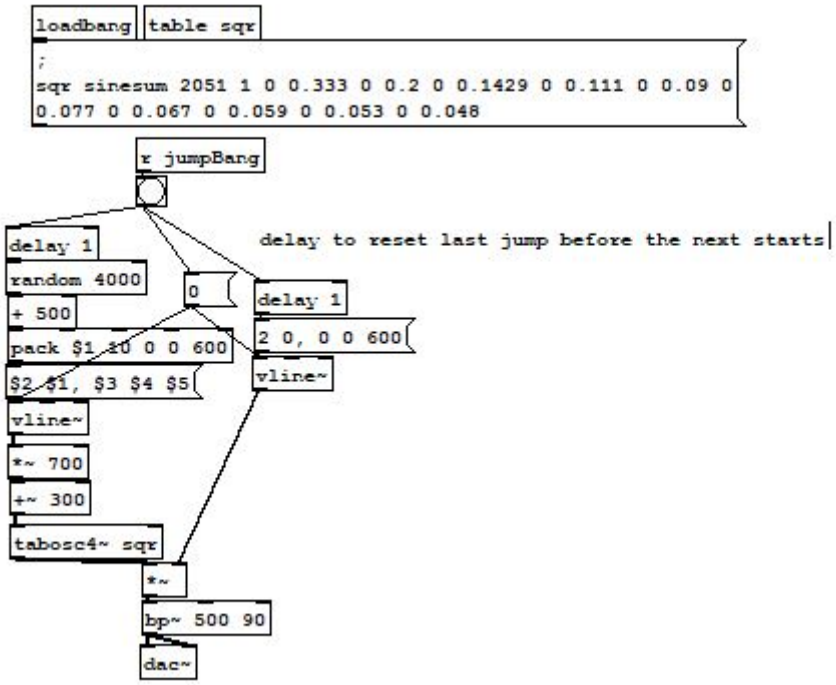

Figure 2: A pd patch that generates a square waveform with rising frequency until it dies out.

3.1.3 Throw

We started with a white noise block and a band-pass filter. When we had figured out our desired center-frequency of the filter we multiplied the output with an envelope to raise and lower the volume with time. The envelope we used is generated with a simple vline block. The time it takes the envelope to rise and decay is something we figured out with testing. Now to sweep the frequency of the noise we also used a vline block. The output of the block we multiply with a number and add it to the base center-frequency of the noise. The base center-frequency is 500 Hz and we gradually sweep it upward to 3500 Hz. This way we were satisfied with the sound of the throw, but so every throw wouldn't sound exactly the same we decided to also sweep the Q-value of the band-pass filter. We started out with a Q-value of 15 and sweep it upward by a maximum value that is between 0.0 and 20.0. This adds a random behavior to the sound. Figure 3 shows the Pd patch that generates this sound. The pd patch can be found at "/Sounds/Pd patches/Santa's activity/throw.pd"

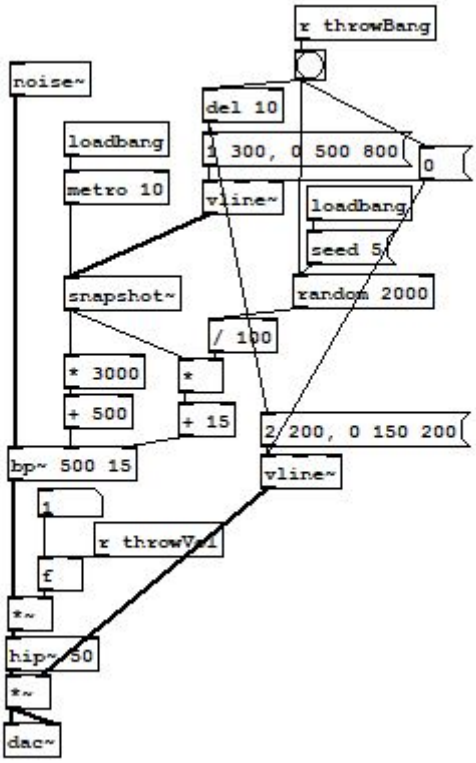

Figure 3: A Pd patch that sweeps the center-frequency of a band-pass filtered noise.

3.1.4 Low Player Health

To get this we made an oscillator at a lowfrequency and added to it another oscillator with a frequency which is slightly offset from the other one. The offset is done with a random floating point number generator. The random number is set to be between 0.0 and 1.0. When the sound from the two oscillators are added together they produce a beating sound. By adjusting the base frequency we got a sound we were satisfied with, but to make it a little bit better we wanted to get rid of the low-frequency noise in the sound. We did that by passing the output through three high-pass filters with a cut-off frequency of 150 Hz. The reason for using three filters is to get a sharper transition at the cut-off frequency. Figure 4 shows the Pd patch we made to generate a low-frequency beating tone to use to indicate when the player has low health. The pd patch can be found at "/Sounds/Pd patches/Santa's activity/low_health.pd"

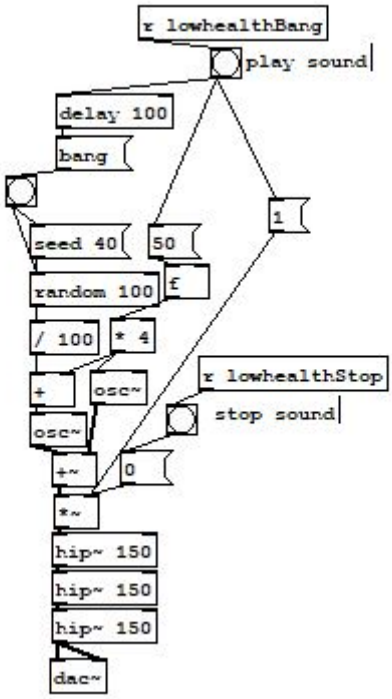

Figure 4: A pd patch that generates a beating tone at low frequency.

3.1.5 Player Death

The sound we ended up with can be seen in figure 5. It starts out with a 50 Hz square wave generated with additive synthesis. The patch then sweeps the frequency of the wave down until it fades out. The pd patch can be found at "/Sounds/Pd patches/Santa's activity/death.pd"

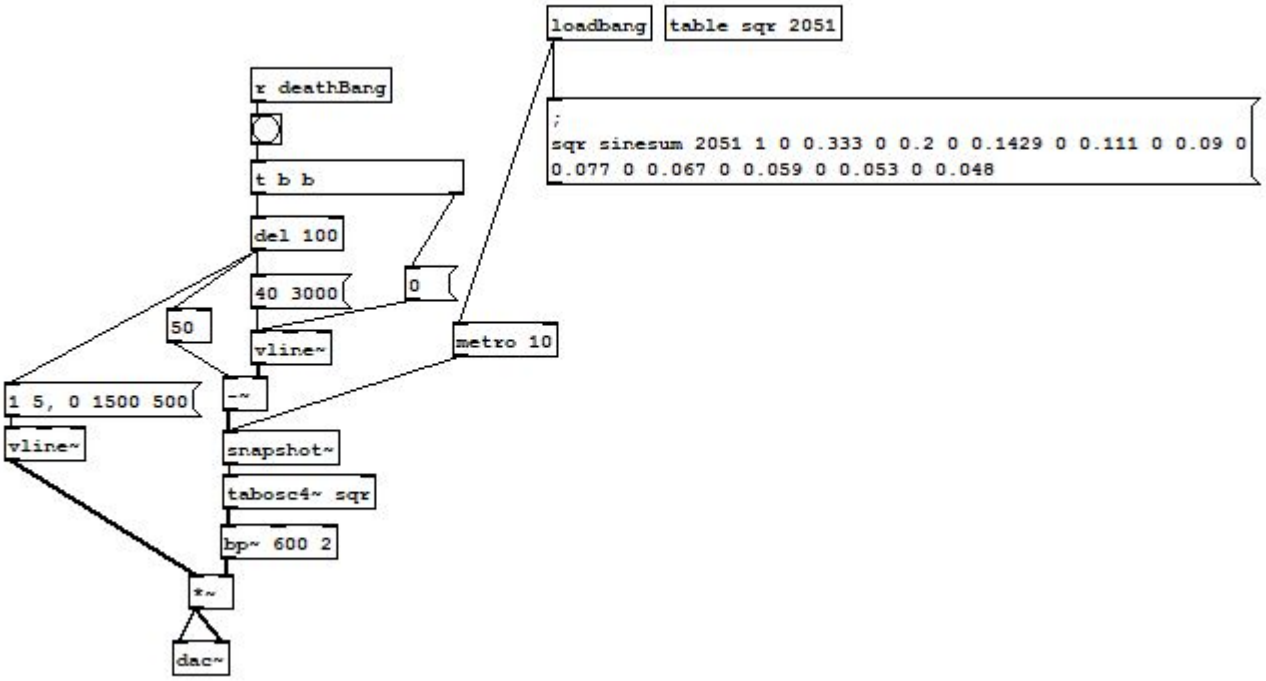

Figure 5: Pd patch that sweeps the frequency of a square waveform.

3.1.6 Player Taking Damage

Although we set out to make a sound effect that sounds like "aahhhh" we ended up by using a different sound altogether. The reason for this is that when we were making the low player health sound effect we stumbled upon a sound that we thought would be a good damage effect. This sound is two oscillators at different frequencies that are not far apart. The distance between them is 11, 14, 24 or 43 Hz. The sound that is generated is different depending on the frequency difference. Figure 6 shows the Pd patch that makes the damage effect. The pd patch can be found at "/Sounds/Pd patches/Santa's activity/damage.pd"

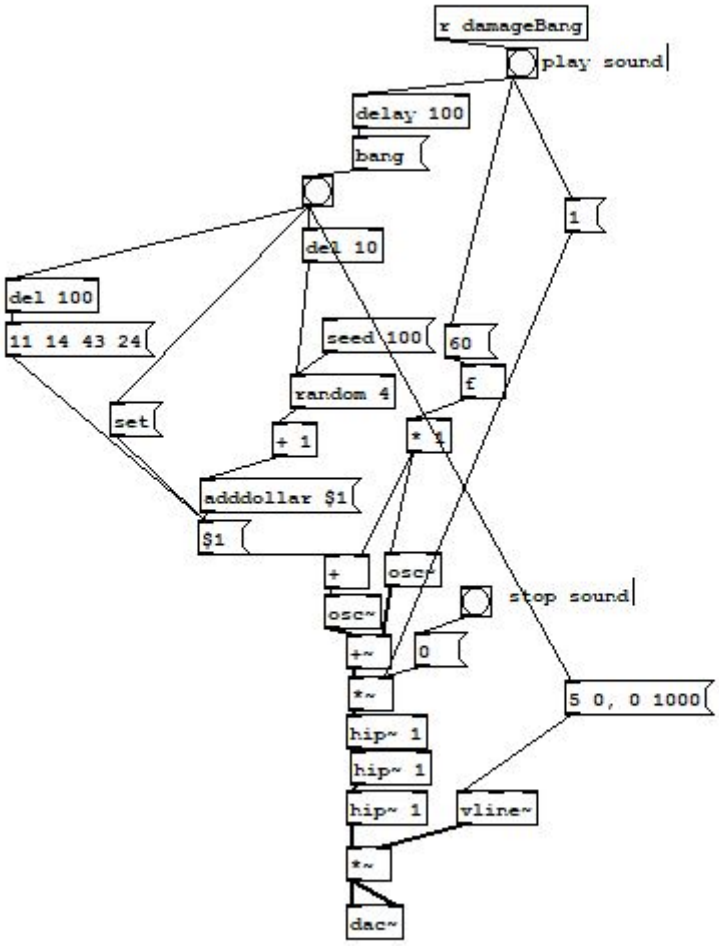

Figure 6: A Pd patch the generates two oscillators slightly out of sync with one another.

3.1.7 Chimney

Smoke

We recollected that noise of a certain color (in a specific frequency range) sound like gas leaking or escaping a container. We thought to our selves that using that would be an optimal start for the chimney blowing smoke. The frequency range of the noise that we used is from 15 kHz to 20 kHz, because we thought the sound it made was appropriate. The next step we took was to make the sound develop with time. We decided that since the chimney blows smoke in intervals, i.e. blows smoke for a few seconds, then rests for a few seconds before blowing smoke again, it would be appropriate for the smoke sound to increase to start with and then decay into nothingness. To do this we used a Pd block name *ead* , which stands for exponential attack/decay algorithm. This blocks starts by exponentially increasing a signal for a fixed number of seconds and then it exponentially decays the signal until there is no signal left. We ended up using two of these blocks to make the smoke sound attack more rapidly. Figure 7 shows the final version of the Pd

patch for generating the sound of a smoking chimney. The pd patch can be found at "/Sounds/Pd patches/Santa's activity/ChimneySmoke_ead.pd"

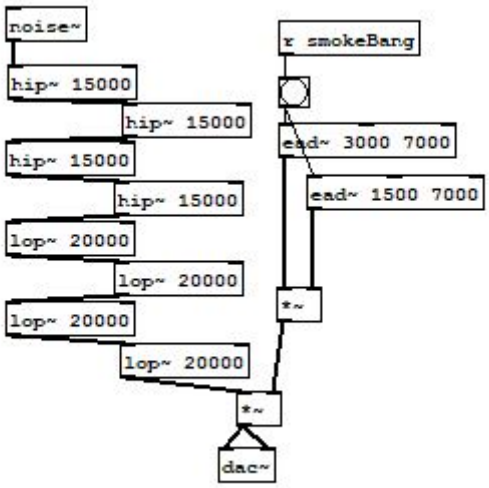

Figure 7: Pd patch for procedural generation of a sound for smoke blowing out of a chimney.

Down

The process we went through to make this sound was actually quite simple since it was by happenstance that we ran into what we considered a good basis for this effect. We were playing around with mixing a few band-pass filtered noises together with different center frequencies and Q-values[7]. When doing so with frequencies between 3 and 7 kHz we accidentally got this effect that we intended to implement later on. Of course, we took this opportunity and expanded a little on this base tone to get a little variety into the sound. What we thought the sound needed was some short bursts of high-frequency tones to make the sound more glistening. We did so by adding a similar element that we already had. We lowered our center frequencies a little bit and shortened the time the sound lived for. Then we added two metro blocks and a random block to trigger these short bursts at random interval while the base tone was playing along. By adding an even further range of short tones the sound could be enriched further, but we decided that would be a part of further sound development. Figure 8 shows the Pd patch where this "magical" sound effect is generated. The pd patch can be found at "/Sounds/Pd patches/Santa's activity/going_down_chimney.pd"

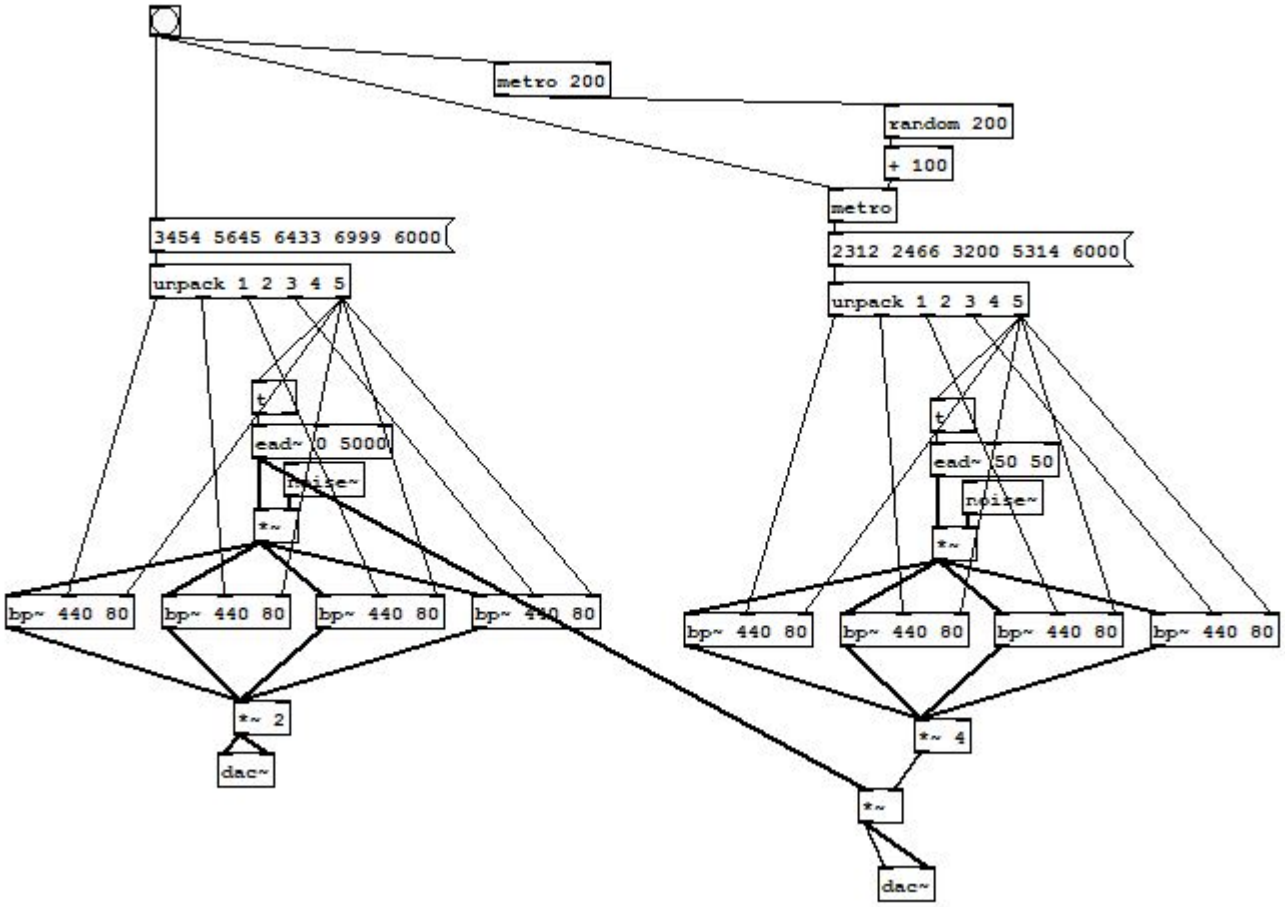

Figure 8: Pd path for procedurally generating a magical like sound effect to use when going down chimneys.

3.1.8 Gift

To implement this sound we decided to try and use additive synthesis. To start with we set up a sub-patch in pd that implemented the additive synthesis. After that, we connected the additive synthesis output to a line envelope. The reason for this is that it let the sound die out with time. When that was done we started to figure out at what frequency we wanted the sound to be at. We quickly realized that by using one base frequency in additive synthesis our sound would be very flat, and in fact boring. To fix this we decided to try to add to our patch two more additive synthesis blocks at higher frequencies. We also changed the envelope affecting the output of each of these blocks. To find the frequencies we wanted for the two new additive synthesis blocks we decided to use trial and error. We tried a few frequencies and in the end were happy with frequencies of 999 Hz, 1500 Hz and 1800 Hz. We also decided that it sounded better when the frequencies were not exited at the same time, so we introduced a delay block. Finally, we added some multiplication blocks to the patch to be able to adjust the ratios between the three base frequencies. Figure 9 shows the final patch for the gift sound. The pd patch can be found at "/Sounds/Pd patches/Environment/gift.pd"

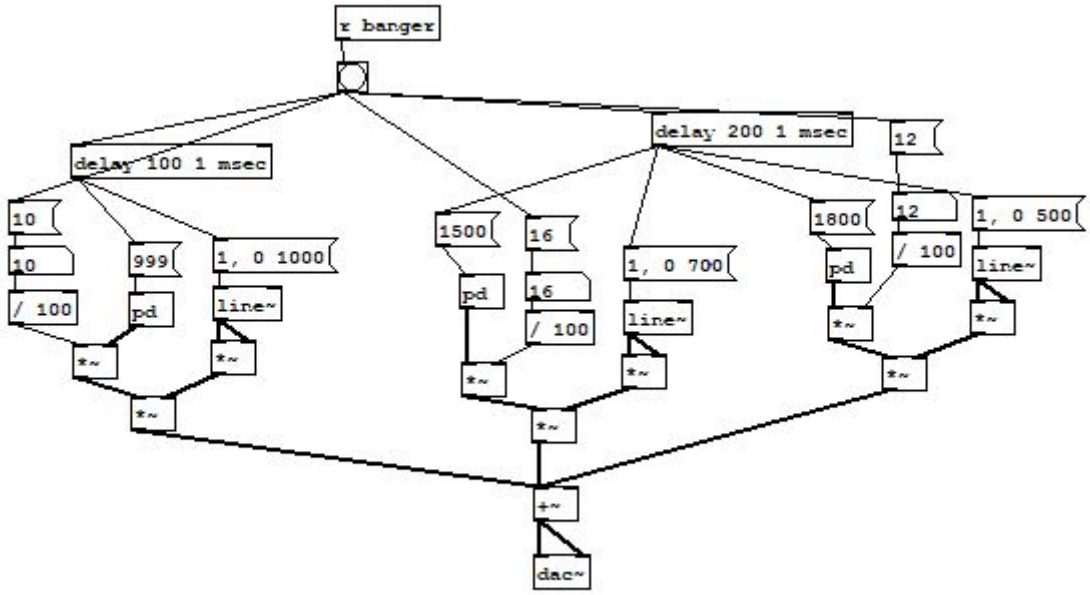

Figure 9: Pd patch to make a "bling" sound for when a player picks up a gift.

The sub-patches on the figure all contain additive synthesis. The additive synthesis used can be seen in figure 10.

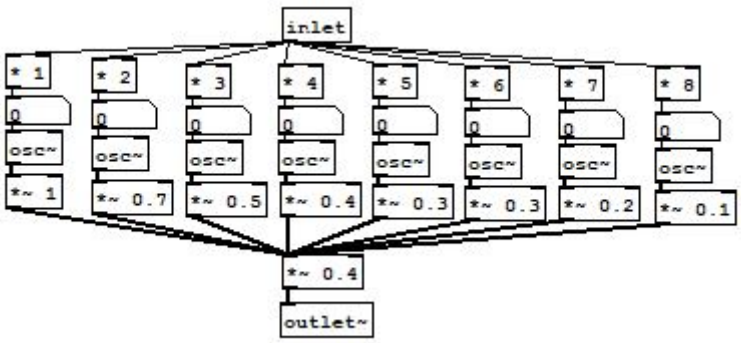

Figure 10: Pd patch for additive synthesis.

3.1.9 Bomb

Fuse

The implementation ended up being really simple since a fuse has a lot of white noise and by doing a high pass cut-off one way and low pass the other and then multiply them together we ended up with pretty close fuse sound. It only plays for 500 ms as the animation in the game isn't long but in the future we would have the bombs go off at different times and to do that we would only have to send in a float number of how long it takes the fuse to burn. The pd patch can be found at "/Sounds/Pd

patches/Environment/fuse.pd"

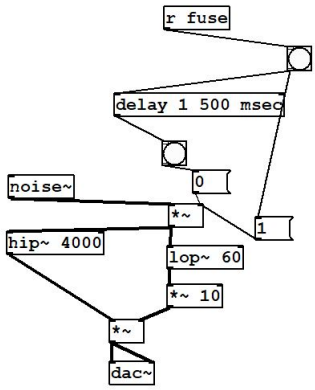

Figure 11: A pd patch that generates sound for a bomb fuse.

Explosion

For the implementation of the explosion sound, we decided it was best to start out with a noise block in Pd and filter that noise to specific frequencies. For the sharp attack sound, we tried a few different low-frequency noises by using a low pass filter in Pd. The low pass cut-off frequency we decided to use was 50 Hz. To get the sound to attack and then decay we used an exponential attack/decay algorithm and played around with the attack and decay times until we were satisfied with the sound. For the low-frequency rumble, we decided to try and use a bandpass filter to get a specific low-frequency range in our noise and add that noise to our sharp attacking noise. The frequencies we ended up using for the rumble range was from 50 Hz to 250 Hz. We also used an attack/decay algorithm on the rumble so the explosion would play out over time. At this point we thought we had a good basis for an explosion, but weren't entirely satisfied with the ratio between the rumble and the sharp attack. To fix this we introduced two multiplying factors into our Pd patch, one that is multiplied by the sharp attack and one with the rumble. To find the right multiplying factors for the explosion we played around with many numbers, but ultimately we settled for multiplying factor of 120 with the sharp attack and -66 with the low rumble. At this point, we were pretty satisfied with the explosion sound but wanted to introduce a little random behavior into it such that it wouldn't sound exactly the same every time. The way we did this was to add to the multiplying factors a random number between -10 and 10. Figure 12 show the Pd patch that was created to generate the explosion sound. The pd patch can be found at "/Sounds/Pd patches/Environment/explosion.pd"

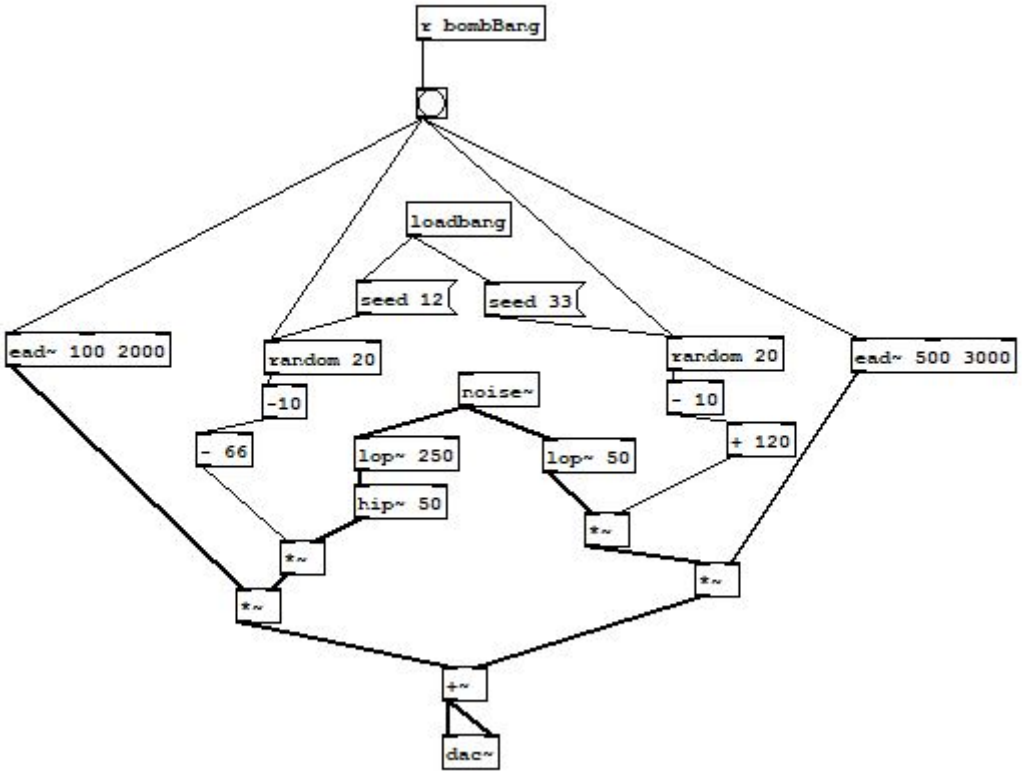

Figure 12: Pd patch created to generate a sound for an exploding bomb

.

3.1.10 Icicle

The implementation tactic for this sound could be described as getting away with doing as little. In sound design, it's often a simple sound that's supposed to describe something complicated. It's debatable if it works each time and that can also be said about this solution. We use the simple material patch from Obiwannabe[7] to get a glass type sound that's played few times in a row very fast to make it sound like glass breaking into pieces. The pd patch can be found at "/Sounds/Pd patches/Environment/icicle.pd"

3.1.11 Wind and snow

For the snow we used the Obiwannabe rain tutorial[4] as base but we felt it was too "wet" sounding so we stripped it down to make it more crunchy. Playing it alone makes it feel a little bit like fire but when played with the wind it comes out nicely. The wind was implemented by having three sources of vline that are fed into bandpass filters with different Q-values. Each vline moves the frequencies up and down over different times to give it a very wind-like sound in a simple manner. The pd patch can be found at "/Sounds/Pd patches/Environment/wind_and_snow.pd"

3.1.12 Milk and Cookies

We decided to change pace with this sound. We recorded some sound of water being poured from a bottle into a jug, and of a partly filled water bottle being shaken. We imported these sounds into Audacity with the intention of using them to make a drinking sound. We extracted a part of the sound and slowed it down by a factor of two. This gave us a nice sound that kind of sounds like someone drinking. We then took the resulting sound and repeated it three times at different strengths (sort of like an echo). When the ratio between each sound and space between them had been fixed the resulting sound was something we felt we could use for our milk and cookies. Instead of using Pd to further change the sound we decided to play it unaltered. The .wav file can found be at "/Sounds/recordings/modified/drinking.wav"

3.2 Soundtrack

3.2.1 Menu song

The approach to a menu song we went with was the classic, a melody looped over and over again. Since our songwriting skills aren't the greatest we looked up one of the most recognizable Christmas melody's, Jingle Bells by James Lord Pierpont, and recorded it with a Ibanez electric guitar and Uteck Guitar-cube [8]. We tried to use Pd to record it but there was a lot of clipping sound from the Guitar-cube so we used Guitar Rig[2] instead. The main melody was recorded twice but in different octaves, so they were merged together in Audacity[1]. Pd patch was created to read the jinglebells1.wav and then we created a tambourine sound to make it more in the holiday spirit.

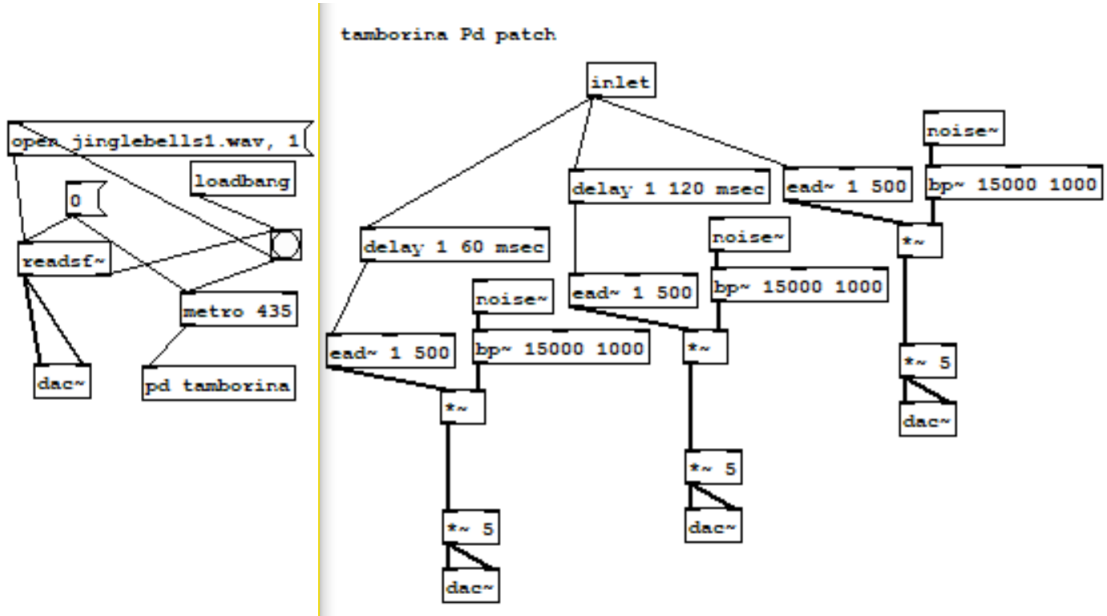

Figure 13: Pd patch that read wav file and plays tambouring type sound

.

3.2.2 Level 3 song

We used the same approach as in the menu and recorded ourselves the melody in Deck the halls on a guitar. This time we didn't add any other sounds to it. It's just a beautiful raw sound of the guitar.

3.2.3 Level 2 song

When we were thinking of things to create in PD a question came up, how hard would it be to create a cover of a song? Turns out, quite hard if you don't know what you are doing but we ended up with a somewhat "cover" of the song Silent Night by Franz Xaver Gruber. How it was accomplished was getting the main notes from the melody and converting them into pitch frequencies[3] and add them to a table in PD and then loop through them with metro. You clearly hear the melody and it has a little bit of retro throwback sound to it but you can't say it's beautiful. We could also have done this by putting midi notes to the array and then let mtof transpose them into frequency. Overall it was a fun experiment.

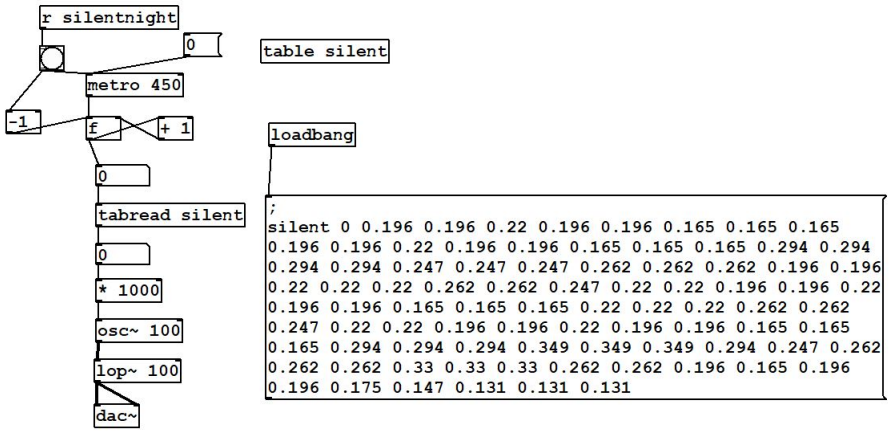

Figure 14: Pd patch that plays the Silent Night melody

.

4 Analysis & Conclusion

After all our sound design was done we decided to listen to our sound effects again and evaluate if they needed any changes. Overall we were pretty satisfied with our product, but in some instances, we would have liked to be able to get better quality sounds or at least something that sounded a little bit more like what we had planned to begin with. An example of that is the death sound, which doesn't sound like we planned at all because we had to make a decision of settling for a different sound in order to get on with our design of further sound effects. With further development we would most definitely improve upon our current work, we would clean up those sounds that sound a little sluggish, re-implement our death sound and make more sound effects and more

music for the game. An example of the sound effects that we would like to add to our game are:

- Add a sound that would play when the mouse hovers over buttons in the game.
- Sound for the scene where the score for the current level is calculated. These sounds would include
  - A counting sound for when the score is scrolled from zero to the actual score in the game.
  - A sound that plays when new items are displayed on the scene, f.ex. if the player reached the high score table. This sound would maybe sound like a "boom".
  - Applause or some other kind of congratulatory sound if the player reached the top 5 high scores.
  - A themed song or sounds that play depending on whether the player finished the level or if the player died.
- Add some sounds to the enemies. The enemy would only play sounds if it is rendered on the screen.
- A sound effect for a melee attack.
- A sound effect for when a snowball or other throw-able object collides with the level or a character.
- More music for the game.
- Any further sound effect that would be needed with further game development.

4.1 LibPd

To play the sound effects and music that we made in our game we used a Pd library known as LibPd. This library allows unity to open and communicate with Pd patches. This process was not as straightforward as we thought in the beginning. To start with it was a little bit hard to get the library to work in Unity. This was because we were using the newest version of Unity which doesn't support a 32 bit editor, but in the LibPd library, we had downloaded needed this 32 bit support. We ended up installing an older version of Unity with 32 bit support and after that, it was easy to understand how to load a Pd patch. Another problem for us was that LibPd does not support the use of Pd externals and certain blocks in patches. As we didn't know this in the beginning, we had used many of these objects to make our sound effects. Some time went into re-making the patches to work in Unity and in some instances, our sounds changed when we did this. Because of the sound changing we decided to make a separate patch for the sounds

that are playable in Unity, this way we still had access to our original sound effects. The version we had of LibPd only had an implementation to load one Pd patch at a time. Instead of wasting time trying to implement multi-patch loading we decided that after we had implemented our sound effects we would copy all the original patches into one patch for the game audio. To trigger each of the sounds separately we made use of receivers that can be triggered by Unity. This patch can be found in the Unity game folder at "Assets/StreamingAssets/PdAssets/GameAudio.pd".

4.2 What we learned

When we started this project neither of us had any experience with sound design, this meant that a lot of trial and error went into our first sound effects. As development got along we felt more and more comfortable with using Pd, and also felt that we were better at analyzing what we wanted our sound to be like. This meant that we started getting better at getting a nice base sound for our effects without as much trial and error. We also learned new ways to make things in Pd, an example is that instead of making a big patch for additive synthesis we learned to do it with a simple table and a table oscillator. We were able to make use of the tutorials we went through in class and modify them or even only use some part of them in our own patches [5].

The next time we develop sound for a computer game we would approach the issue differently because of what we have learned during this development process. It would be easier for us to brake sounds down into separate components, but we understand that to get even better at it we would need to listen to the sounds in our environment more analytically and we would need to spend some time playing around with different methods of generating sound. With such practice we are sure that we would be better equipped for better sound designs. Another thing we would make use of is to record different sounds, and start a sound library. We could then play around with modifying these sounds to get some nice sounding effects that are usable in computer games.

References

[1] Audacity. http://www.audacityteam.org/, 12 2017.

[2] Guitar rig. https://www.native-instruments.com/en/products/komplete/guitar/guitar-rig-5-player/, 12 2017.

[3] Pitch to frequency mappings. http://peabody.sapp.org/class/st2/lab/notehz/, 12 2017.

[4] Rain tutorial. http://www.moz.ac.at/sem/lehre/lib/pd-sounddesign/tutorial_rain.html, 12 2017.

[5] Sound design. http://www.moz.ac.at/sem/lehre/lib/pd-sounddesign/index.html, 12 2017.

[6] Super mario sound effects. https://forum.pdpatchrepo.info/topic/10396/super-mario-sound-effects-reproduction, 12 2017.

[7] Tea tutorial. http://www.moz.ac.at/sem/lehre/lib/pd-sounddesign/tutorial_tea.html, 12 2017.

[8] Uteck guitar-cube. https://www.aliexpress.com/item/Uteck-Guitar-Cube-ASIO-USB-audio-Interface-DI-fit-for-soft-Guitar-Rig-JAMVOX-AmpITube/1984116392.html?aff_platform=aaf&cpt=1513152598029&sk=zj6qB6AIM&aff_trace_key=7808f7d0790942aea496da0aa4ee33df-1513152598029-04109-zj6qB6AIM&terminal_id=2b19f40a91344e17b633c78316d4c506, 12 2017.