Generating 8-Bit Sound Effects Using Interactive Evolution

Tobias Garpenhall

Game Development
Bachelor's Thesis
15 ECTS credits
Spring 2022
Supervisor: Steve Dahlskog

Generating 8-Bit Sound Effects Using Interactive Evolution

Tobias Garpenhall
Game Development
Malmö University
Malmö, Sweden
tobiasgarpenhall@gmail.com

***Abstract*—Interactive evolution is explored and applied for the automatic generation of 8-bit sound effects (SFX). Procedurally generating a type of content can result in greater accessibility, cut development costs, and more. However, a natural problem that follows this approach is user fatigue. An 8-bit SFX generator is developed, tested, and then evaluated to understand its capabilities, usability, and effectiveness of several applicable solutions for reducing user fatigue. Results indicate that the software is intuitive to learn and use while providing a decent variety to the generated content with a probable feeling of progression. The implemented solutions for abating user fatigue show promise towards making the software practically viable. However, there are still areas in the developed artifact that suggest for further study.**

***Index Terms*—Procedural Content Generation; Interactive Evolution; Sound Effects; User Fatigue; Commodore 64 SID chip.**

I. Introduction

In game development, there is a desire to automatically generate game content for reasons such as: cut development costs, improving replayability in games, and accessibility for new developers. [1]. This area is commonly known as procedural content generation (PCG) and a famous example of PCG is automatic terrain generation in Minecraft [2]. There are several methods for generating the game content, and one of them is through a search-based approach that primarily utilizes evolutionary computation (EC) [3]. EC evolves the content progressively until it reaches some desirable quality. The problem with it, however, is when the quality of the content is a matter of subjective opinion and cannot be defined without making certain assumptions. A possible solution to this problem is to allow the user to steer the evolution through what is called interactive evolution (IE) [4]. In IE, the user selects the individuals that they favor, and the algorithm optimizes the output towards their preferences. A frequent application of IE can be found in the automatic composition of music and sound synthesis. That may be because of how the quality of music or sound is commonly expressed through subjective interpretation. However, IE has a flaw; as the process is no longer fully automatic, and a human user has to regularly give input, it may lead to user fatigue [4]. User fatigue can emerge when the evolution is too slow, output is poor or worsening, user is presented with too many choices, etc.

Kaliakatsos-Papakostas et al. [5] developed an IE system that evolves 8-bit melodies using genetic programming and addresses user fatigue by letting the user control the randomness of the evolutionary process. Tokui and Iba [6] apply IE to generate compositions of rhythms and attempt to abate user fatigue by implementing several assistance methods for the evaluation step. The methods assist the user by breeding larger populations and showing only individuals with a higher fitness value calculated by rhythm length and an artificial neural network (ANN) that learns from the users' preferences. Jónsson et al. [7] paper presents a technique for producing timbres by developing a special type of ANN called compositional pattern producing network. They conclude, however, that the method could use improvements in relation to user fatigue by implementing, for example, starting from other peoples' work. Dealing with user fatigue and making the developed artifact useful for practical applications is a common problem to encounter that can be approached in many possible ways.

While music has been actively researched, sound effects (SFX) have gone largely unexplored in the domain. The direct creation of SFX can be a complex process that sometimes requires expensive or complicated tools, making it inaccessible to many. There is a lack of tools that tries to solve this problem by employing a simple and intuitive interactive process for producing a variety of SFX. Creating a practical tool that can make the development easier for a type of content would be beneficial for many. For example, *PicBreeder* [8] works by displaying several similar but unique images and letting the user pick amongst them to further evolve. This simple and intuitive process can lead to the creation of distinct and interesting images. SFX, however, encompasses a vast scope of variance that makes it challenging to start solving. To make the development easier in this study, sound quality is limited to 8-bit which refers to the sound qualities of the 8-bit generation of consoles and computers, and not to its actual specifications.

The purpose of this work is therefore twofold, but it comes together to primarily gain insight into the following research question:

- How does the user experience the usability of a software that employs an IE process for the generation of 8-bit SFX?

The research methodology this study follows is based on

the design science (DS) paradigm [9]. This methodology is applied for developing, testing, and evaluating an artifact that employs IE to generate 8-bit SFX. Additionally, several potential solutions for preventing user fatigue are implemented and examined in the artifact. To produce 8-bit sound quality, an emulation of the Commodore 64 SID chip is utilized. Lastly, evaluation is conducted through multiple qualitative usability tests on the artifact in a controlled environment with a participant, combined with an analysis of its expressive range.

The developed artifact may be used practically for easily generating 8-bit SFX that is accessible to even those with limited experience in the creation of music or SFX. It may also be used to gain insight into the effects of several implemented solutions for reducing user fatigue relevant to the domain.

II. Background Material

This section introduces previous research related to this study and describes the following concepts: procedural content generation, mixed-initiative PCG and evolutionary computation. Lastly, the expressive range, Commodore 64 SID chip, and BASIC are explained.

A. Procedural Content Generation

Procedural content generation (PCG) is defined by Togelius, et al. [10] as *"... the algorithmical creation of game content with limited or indirect user input"*. Furthermore, Togelius et al. [1] defines game content as pretty much everything within a game except for the game engine itself and non-player character behavior. The common reason for using PCG is to remove the requirement of having a human create the content manually. Such a process can become arduous to perform when the quantity or quality of desired content is high, which can lead to considerable expenses in costs and time. The ability to more conveniently generate game content also makes it much more accessible to new game developers, for example: developers wanting to create sound effects for their game but have no experience doing so.

B. Mixed-initiative PCG

PCG is generally an automated process that does not require considerable input from the user to function. When the process does instead require human input to generate the content, the method is known as using mixed-initiative PCG [4]. How much the computer or a human takes the initiative in the process differs from each approach. On one side of the spectrum, there is computer-aided design, which is when the computer assists the human in their creative process. On the other side, there is interactive evolution (IE), that is when the human guides the generator to create the content they prefer.

An example of IE is *PicBreeder* [8] where they create and evolve images using a compositional pattern producing network (CPPN). PicBreeder starts by showing a simple set of images that the user can select to either mutate or save. When mutated, a new set of images are shown that have been slightly altered from the original one. A unique feature of *PicBreeder* is the ability to start from other peoples' work, which is one of the inspirations used in this study for reducing user fatigue.

C. Evolutionary Computation

Evolutionary computation (EC) is a stochastic search-based algorithm inspired by biological evolution [3]. EC starts by creating a population of individuals, where each is then evaluated and assigned a fitness score. The individuals with higher fitness have a greater chance of producing offspring, while individuals with lower fitness may be removed. During reproduction, it is possible for offspring to be mutated that makes some random alterations to the gene. The problem in EC is defining a well-performing evaluation function that accurately measures the quality of the content. A solution to this problem is to allow the human to drive the evolution by evaluating the content themselves, which is known as interactive EC (IEC). Takagi [11] presents an extensive review of numerous research papers related to IEC, while also examining user fatigue that is a prevalent problem in the area. There are many ways of combating user fatigue, for example, Kamalian et al. [12] attempt to reduce the required number of human evaluations in the interactive process by developing a function that predicts the users' evaluation score.

IEC has been applied to a number of domains where the quality of the content is difficult to objectively measure. For example, Cardamone et al. [13] developed an algorithm that evolves racing tracks based on the user's assessment; Hastings et al. [14] applied an IEC tool called NEAT particles to interactively evolve particle systems via user evaluation. Hoover et al. [15] introduced a multifaceted game content generator that is explored through a game called *AudioInSpace*. While the game is being played, data about the player's preferences are being collected implicitly. The data is then used to rate and evolve the CPPNs that generate the visuals and audio.

For EC to function, the content needs to be represented in the form of a genotype [3]. A genotype is the collection of data used to represent an individual during the evolutionary process. The reason for having to define a genotype is to more easily classify and evaluate individuals, while also being computationally efficient. The genotype is then converted to a phenotype, which is the actual content.

A common problem in EC is premature convergence that occurs when the evolution becomes stuck with suboptimal individuals. Many methods help solve this problem by, for example, increasing population size, incest prevention [16], and modifying the parameters of the algorithm.

D. Expressive Range

Expressive range refers to the extent of varied content that the generator is capable of producing [17]. Evaluating the expressive range is important to understand the capabilities and quality of the generator. The typical method of gathering the expressive range is through measuring the content in certain metrics such as linearity and leniency and then mapping it to a heatmap. Smith and Whitehead [18] evaluate the quality

of a game level generator by generating multiple levels and comparing them by their linearity and leniency. They define linearity as "the 'profile' of produced levels" and leniency as "how forgiving the level is likely to be to a player". It is important that the chosen metrics are easily understood and suitably characterize the content so that a qualitative analysis can be made on the generator.

E. Commodore 64 SID chip

The Commodore 64 SID chip (MOS6581) is a sound interface device (SID) capable of creating various 8-bit sounds [19, 20]. The SID chip consists of three voice synthesizers that can be used separately or in parallel with each other. Each voice has a number of parameters that can be set, which includes: waveform, ADSR and filters. Volume can furthermore be set, but is global for all voices. Waveforms affect the shape of the sound, and the SID allows for setting several distinct types: sawtooth, variable pulse, triangle, and noise. The volume contour for each voice can be altered by the ADSR generator, which includes: attack, decay, sustain, and release. Filters can be used to strengthen or cut some frequencies. SID also offers complex features such as sync- and ring-modulation effects when two waveforms are combined. A detailed description of all parameters is given in chapter 4 of *Commodore 64 Programmer's Reference Guide* [21].

This study will make use of an accurate emulation of the Commodore 64 SID chip called *reSID* [22]. The library contains several notable functions, e.g., write, enable filters, and audio sampling. *Write* alters the parameters through a register when specifying an offset and value. Modifying this register allows for a number of distinct sounds to be generated.

F. Commodore 64 BASIC

To interact with the SID chip in the Commodore 64, the user has several commands available for usage [23, 21]. The most vital command when creating sound is *POKES+X,Y*, which sets the value *Y* to *S+X* in memory. *S* is the memory location for sound synthesis; this essentially alters the parameters of the SID chip, for example, *POKES+1,Y* adjusts high frequency for voice 1. Another important command is *FOR* that executes a block of instructions multiple times. *FOR* loops can be used for audio sampling in the Commodore 64. Finally, the *RUN* command is specified that executes all the prior entered instructions. These commands are used as inspiration in this study for writing a custom interpreter that executes simple instructions in a text file.

III. Method

A. Design Science

This study follows the design science (DS) [9] research methodology. The goal of the DS process is to successfully carry out the development, testing, and evaluation of solutions for practical problems in engineering and computer science disciplines. Adhering to a well-established methodology helps validate the outcome of the study and recognize the steps that were taken to reach that outcome.

B. Artifact

A software was developed that is capable of generating various 8-bit sound effects (SFX) by using an emulation of the Commodore 64 SID chip[1]. The user is presented with a large assortment of SFX that they can select to either evolve or export. The selection marked for evolution are inputted as models into an algorithm that attempt to produce a population that are similar but slightly different from the provided models. The resulting population from the algorithm is what is then presented to the user for further evaluation.

Several quality factors were taken into close consideration when developing the software towards reducing user fatigue, i.e., effectiveness, efficiency, and understandability. This includes: implementing several assisting tools, providing an intriguing starting point, designing a simple and minimalistic user interface (UI), etc.

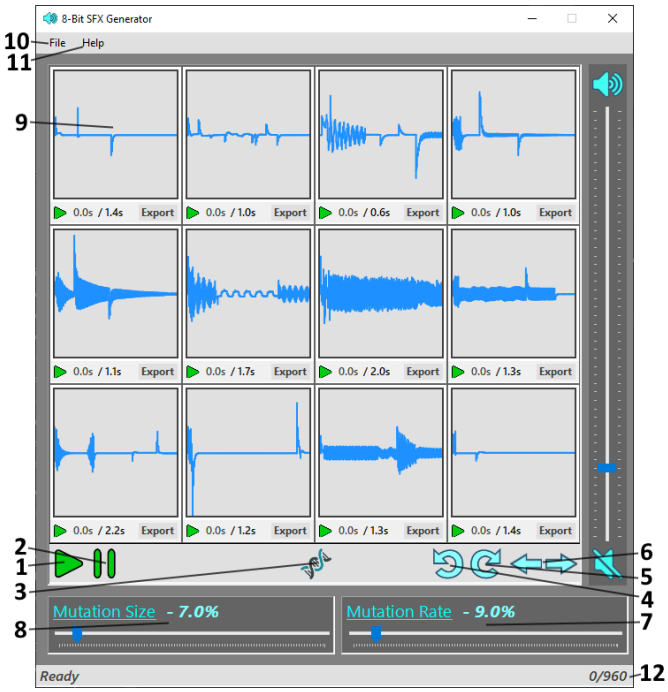

Fig. 1: The UI of the artifact with numbering

The user interface is shown in figure 1, containing an overview of the generated content and assisting tools. All tools have a tooltip and an associated image with it to provide hints for what it does. Some tools are explained in the following list:

- ***Global player (1):*** Starts playing the SFX incrementally from top left to bottom right. Pressing the button again while already playing will advance it to the next sound. Designed to allow for a more relaxed experience, allowing the system to play the SFX for you.
- ***Pause (2):*** Pauses the player.

[1]https://github.com/Daikalos/8Bit-SFX-Generator/releases/tag/v1.0.1

- ***Evolve (3):*** Executes the algorithm, inputting the selected SFX as models. If no SFX are chosen, the algorithm cannot be executed.
- ***Reset (4):*** Completely restarts the process, starting from scratch. Useful for generating different starting points if the current one is poor.
- ***Retry (5):*** Brings you back to the previously shown population. Can only go back one iteration and will not work if there is nothing to return to. Being stuck with the outcome from the algorithm, a process the user cannot fully predict due to its innate randomness, will impact the experience negatively when it is poor. Therefore, retry was implemented to prevent this problem.
- ***Move (6):*** Shows a different subset from the current population. Moving also clears the current selection of models. The move was implemented to prevent being restricted to a select few choices and expand the available options.
- ***Mutation rate and size (7, 8):*** These parameters make it possible to control the randomness of the process. Smaller values result in the generated sounds being less varied and vice versa. Mutation rate modifies how many of the created offspring are mutated, and mutation size modifies the magnitude of which each respective offspring is mutated.
- ***Help (11):*** Shows a window that describes the purpose of the software and the containing tools.
- ***Subset (12):*** The current subset is shown on the right of the bottom status bar together with the total viewable population. The entire population is chosen to not be viewable to prevent possibly overwhelming the user with too many choices.

The SFX are shown in 12 smaller windows **(9)** at a time that contain a waveform visualizer, length in seconds, play, and export. The visualizer enables the user to predict its contents so that quick and basic decision-making can be made if the sound is interesting or not. These windows can be clicked to select or deselect the corresponding sound effect for being used as models for the algorithm.

The software features the ability to save and load SFX and populations **(10)**, but only in the format of a *.txt* file. When saved, the genotype of every individual in the current population is saved into a *.txt*, where each is separated by a *RUN* command. A specific sound effect can also be saved similarly by exporting. Loading reads the selected *.txt* file and interprets each segment in the file that is separated by a *RUN* command as an individual. If the size of individuals is less than a certain threshold, it uses them as models and executes the algorithm. Otherwise, the individuals are directly copied into the population and outputted. Example populations and SFX can be found together with the executable, and if the user desires to, it is possible to directly modify the genotypes in the *.txt* files and even create new ones.

The initial population was at first randomly generated, however, it rarely produced any interesting SFX. The starting point would have to be improved to prevent this problem. The solution was to explicitly define a collection of interesting SFX and pick randomly among them to use as models for the algorithm that executes on startup or reset. This results in a longer loading time before the software is usable, but it leads to a more engaging starting point.

C. Algorithm

The algorithm that the software utilizes is based on the standard genetic algorithm (GA) methodology that involves several notable steps: initialization, evaluation, selection, crossover, and mutation. A flowchart of the algorithm can be seen in figure 2.

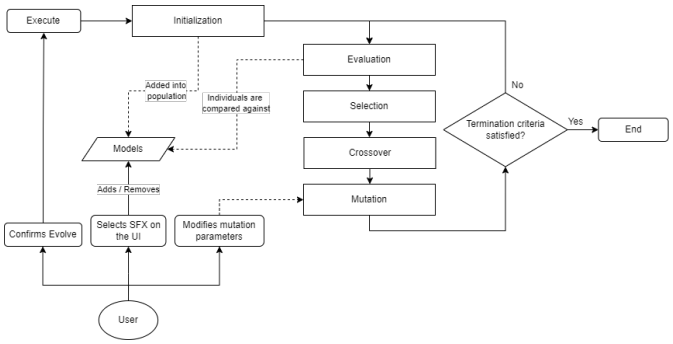

Fig. 2: Basic flowchart of the algorithm

But for the algorithm to work, the content needs to be represented as a genotype. The genotype representation that was chosen is a sequence of simple commands that can be either *'poke X Y'* or *'sample X'*. *Poke* sets value *Y* at offset *X* in the SID's register, and *sample* generates X number of audio samples. Conversion to phenotype is accomplished through an interpreter that reads and parses each line in the genotype from top to bottom. The following portion is an example of a genotype that produces a bell sound when converted to a phenotype.

```
poke 24 14
poke 1 130
poke 5 9
poke 15 30
poke 4 21
sample 1000
poke 4 20
sample 1000
```

Initialization begins by randomly generating a population. To avoid redundancy, an individual is created by picking a random amount of *samples* that are assigned a random number of *pokes*. The number of *pokes* before a *sample* is less than the size of parameters that can be set in the SID's chip. Because repeating the same offset would make it redundant and computationally inefficient for the interpreter. Furthermore, the inputted models are copied into the population at initialization, which nearly

guarantees that similar individuals will be generated at a faster rate. A possible downside to this is that some variation can be lost.

In evaluation, each individual is assigned a fitness score. The fitness is calculated by comparing how similar the individual is with each of the provided models separately. A fitness score of 1 is given if the individual is 60% similar to a particular model, and 0 means no similarity at all. Anything above 60% returns a diminishing score. This fitness function means, for example, that the fitness can become proportional to the number of selected models but is unlikely to occur if the provided models are different. Other fitness criteria include the length of the audio, if it is too long or too short, it is given a negative score.

In specifics, the similarity is calculated by first dividing the genotype of both the individual and model into segments between every *sample* and defining a score variable set to 0. Then, the offset and value in every *poke* in the individual is compared against every other *poke* in the model in the corresponding segment. For example, the first segment in the individual is compared only against the first segment in the model and so on. If a poke with the same offset appears multiple times in a segment, only the latest one is used, while the others are ignored. When it has successfully found a pair of *pokes* to compare, it adds to the score given by the following equation:

$$1.0/(0.33 * |v0 - v1| + 1.0)$$

$v0$ is the value of the *poke* in the individual, and $v1$ is the value of the *poke* in the model. Similarly, the value of the *samples* in the end of the segments are also compared against each other, given by the following equation:

$$1.0/(0.01 * |v0 - v1| + 1.0)$$

The auxiliary values were chosen through trial-and-error on what yielded the best results from the algorithm in relation to its intended objective. The score is finally divided by the number of commands in the model's genotype, which is referred to as its size, to normalize the value between 0 and 1 and is then used to determine the fitness as mentioned earlier.

The selection is rank-based [24] where the individuals are selected based on their rank in the population. Rank is determined by sorting the individuals by their fitness in descending order. The best has a higher chance of being selected and vice versa. This method helps avoid a bias toward outstanding individuals and reduces the chance for premature convergence. The individuals that have been selected are called the *elite* and can be chosen for reproduction during the crossover. The remaining individuals are removed from the population.

Crossover is where two different individuals from the elite are chosen at random for reproduction. The implementation works by creating two identical copies of the respective parents and performing an N-point crossover between the created children. N-point crossover picks N random points on the gene and swaps the contents between the points. Therefore, it can capture multiple segments in a genotype, leading to more diversified children. The children are then added to the population. This process continues until the population reaches its original size. To maintain the variety and further avoid premature convergence, incest prevention [16] was implemented that prevents similar individuals from reproducing.

Finally, each child has a chance of being mutated that is determined by the mutation rate. Mutation size then affects the number of random points on the gene that are picked. Each selected point has its value altered by a slight random amount. There is also a chance, at the point, that a command may be added, removed or flipped, e.g., *poke* becomes *sample* and vice versa.

These steps, except for initialization, run in a loop until either the number of generations or the average fitness of a number of individuals from the population has passed a certain threshold. When these termination criteria are satisfied, the population is sorted based on the fitness of the individual, which allows the user to navigate over the population in a more predictable way. The first individuals in the population have a high similarity to the selected models, while the last is dissimilar.

Whenever the algorithm is executed, the user is unable to interact with the software to prevent certain conflicts that may occur until it is completed, except for minimizing or exiting the application. The algorithm utilizes multithreading to maximize the performance and also to be able to provide feedback of its current progress to the user. The label at **(12)** is replaced with text that shows its progress in percentage in terms of how close it is to meeting the termination criteria. When it is complete, the label is restored to its original state.

D. Evaluation

The evaluation was conducted through multiple qualitative usability tests on the artifact in a controlled environment with a participant, combined with an analysis of its expressive range.

Usability Tests

Usability testing is meant to evaluate various aspects in the software when placed under certain conditions. A user is given the software and is then prompted to perform certain tasks from a facilitator. While this is active, the user's behavior is observed by the facilitator, and then they are asked a series of questions relating to their experience to collect important feedback. In this case, the observations and feedback will aim to be extracted and analyzed from on qualitative viewpoint.

The participants for the usability tests were selected based on accessibility to reach out to and relevancy to one of the target audiences that this artifact is designed for, i.e., people who desire to create SFX for their video game. Therefore, the participants consists of people that have experience in game development in the form of a three-year education in a game development program.

The usability tests consisted of four simple steps that were conducted by the author of this paper. During the second and third step, the tester was also being observed by the facilitator.

1) The participant was advised to read the help section that explains the purpose of the artifact and what the different tools do. Otherwise, they were not allowed to ask questions relating to the artifact during the testing. The testing procedure is meant to simulate the environment where a user attempts to interact with the application for the first time without any outside help. The questions the tester has on the artifact are instead later brought up and discussed in the interview and collected as feedback on elements that may be, e.g., perceived as confusing.
2) The tester was given the artifact to freely use until either satisfied or the time limit is reached, which is set to 10 minutes. The tester is made unaware of the time limits to prevent possible stress.
3) They were then requested to find a simple sound effect that can be either explosion, laser, or coin pickup, until either satisfied or the time limit is reached, which is set to 5 minutes. This helps evaluate the generator's capabilities to assist the user in the process when searching for a specific sound. The sound effect given to a participant is chosen at random to explore cases where the algorithm may have a bias towards creating certain types of SFX.
4) An interview was organized with the tester where they were asked several questions relating to their experience using the artifact.

As shown in figure 3, the questions are aimed at collecting data on what the testers feel about different aspects of the artifact. Follow-up questions may be asked depending on the answer, which is not present in the list. For example, if the participant answers only 'yes', then they may be prompted to answer why. The figure demonstrate the foundation for the nature of the questions that the interview later expands upon, where the participants have the chance to explain their reasoning for the purpose of collecting qualitative data.

1.	Did the tool feel intuitive and easy to use?
2.	Was the output interesting?
3.	Did the process feel progressive?
4.	Was the starting point interesting?
5.	What tools did you feel most useful during the process?
6.	What tools did you feel least useful during the process?
7.	Could you imagine yourself ever using it for practical usage?
8.	What improvement would you like to see?

Fig. 3: Interview questions in order

Expressive Range

The expressive range was gathered by measuring the generated phenotypes in sample size and volume. Sample size refers to the number of audio samples in the phenotype that corresponds to the length of the audio, e.g., 44100 audio samples are equal to one second of audio. Volume was measured by calculating the root-mean-square (RMS) of the samples and then converting it to decibel. RMS is given by taking the square of each sample, summing them together, dividing by the total number of samples and taking the square root of the result. The result was then converted to decibels relative to full scale (dBFS) to be more comprehensible. Relative to full scale means that the samples are compared to the maximum available peak. For example, a sound with 0 dBFS sits at 100% of the maximum level, -6 dBFS at 50%, and absolute silence is at -90 dBFS. The dBFS can therefore roughly be used to discern the volume of the sound. These metrics are then converted to a scale from 0 to 1, and the corresponding position on the array is iterated by one. This array is then converted to a simple 2D heatmap, where red shows a high density while blue is low.

IV. Result

A. Expressive Range

The expressive range can be seen in figure 4 where the decibel relative to full scale is shown on the vertical axis and sample size on the horizontal axis. The horizontal axis applies to each image separately. It was gathered by executing the algorithm anew many times, and for each iteration, all the parameters were randomized except for population size and mutation. At the end of every iteration, a select few individuals from the generated population are converted to a phenotype, measured, and added into a heatmap.

B. Usability Tests

The tests were organized individually, either on-site or online, with a total of 4 testers participating. If conducted online, they were given the artifact to use on their personal computer and asked to share their screen. If conducted on-site, it was done on a pre-prepared laptop that the tester was given.

Each participant is given a unique ID corresponding to the chronological order they were conducted in, for example, the first participant has the ID of 1 and so on.

Setup

Before the tests, the algorithm was configured with a population size of 1800 individuals. The termination criteria are met when either the algorithm has reached 80 generations or the average fitness of the 512 first individuals in the population, sorted by fitness in descending order, reaches a value of 3.

Performance

3 out of 4 tests were conducted online instead of on-site. Despite the different computer configurations being used, the algorithm performed quickly, with an average speed of 2 to 5 seconds taken from execution until completion. None of the participants commented on the algorithm feeling slow in the interview.

Observations

During the second and third steps, the tester was observed to

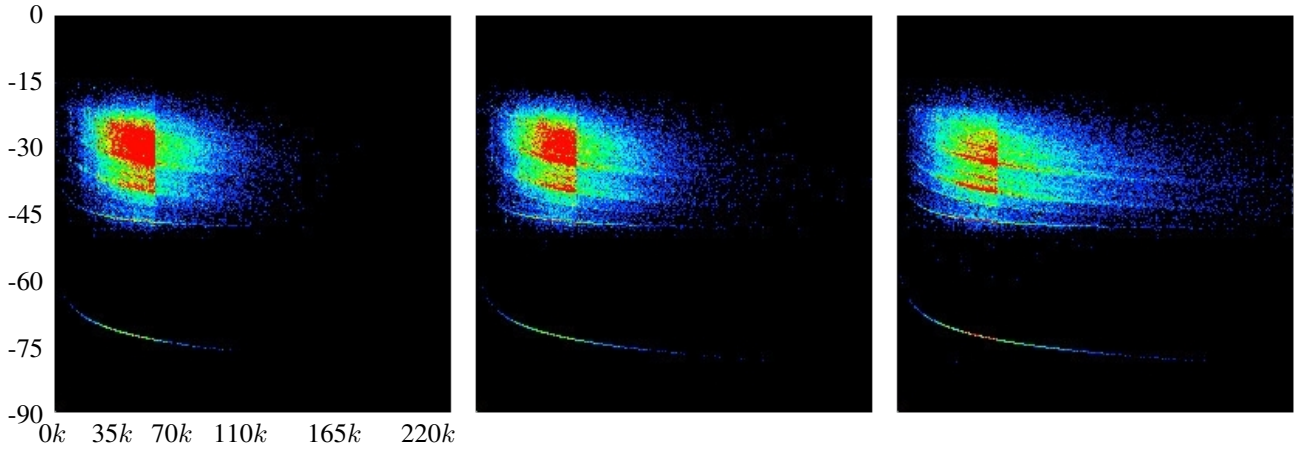

Fig. 4: The expressive range with both mutation parameters set to 0%, 40%, and 90% respectively from left to right

spot certain patterns and peculiarities. Common patterns that were found among the participants include:

- Not listening to all the presented sound effects (SFX).
- Listening to the SFX individually instead of opting to use the global play tool.
- Often not resetting to get different starting points.
- Never using save or load.
- Changing the mutation parameters and usually deciding for a higher value.
- Initially semi-randomly selecting multiple SFX to evolve that became more focused with time.
- It roughly took 10 to 30 seconds in each iteration to select the models and then evolve.
- In the second step, all finished before the time limit of 10 minutes was reached. They roughly lasted between 3 and 6 minutes before the tester felt satisfied.

In the third step, each tester was prompted to find a specific sound effect:

- Tester 1: coin pickup.
- Tester 2: explosion.
- Tester 3: laser.
- Tester 4: explosion.

The testers started to more frequently interact with the tools, particularly the move function and mutation parameters. The selection got more focused on SFX that sounded somewhat similar to the assigned sound effect, or at least what the tester thought would lead towards it. This means that the time it took for each iteration of assessment became somewhat longer than in the previous step. 2 out of 4 started to modify the mutation parameters more often. 3 out of 4 were using move more often than in the previous step. Finally, only 1 out of 4 were able to find an adequate sound within the given timeframe. Tester 1 was able to find a decent coin pickup sound within 1 to 2 minutes, while the others were unable to find their assigned SFX due to giving up or reaching the time limit. Tester 1 used the reset function and was able to find a suitable starting point that quickly lead to the assigned sound.

After the third step was completed, the interview was conducted with the tester.

Outcome

1) Intuitiveness: All the testers agreed that the artifact felt intuitive, “It was easy to use and easy to understand how to get to places” (Interviewee, 1). All were quick to learn the intent of the artifact and how it could be interacted with to generate SFX after reading the help section. Adding tooltips and images for the tools made it easier to identify and understand its function, and showing a preview of the SFX resulted in quicker assessment. Otherwise, there was only a reoccurring confusion on how the mutation rate and mutation size differ from each other. Most likely due to simply missing a description in the help section of how the separate parameters function. There was also a slight confusion on how the move tool worked from one of the participants.

2) Generator’s capabilities: The testers were asked if they felt the generated SFX had a good variety to it while also sounding distinct and interesting. Answers were varied, e.g., “There was a lot of variation” (Interviewee, 1), “I could imagine more variation” (Interviewee, 4), “The SFX could be a bit more melodic” (Interviewee, 4) and “A lot of sounds only had a dip” (Interviewee, 3). 2 out of 4 had a positive outlook on the generator, while the others had more mixed opinions. Some SFX were also experienced to contain undesirable traits, e.g., very similar, no sound at all, and loud pitches or dips. This issue was partly circumvented according to the testers by moving through the population to find other candidate SFX.

Regarding the starting point, where the generator starts from either on startup or reset, “It felt that some sounds were near-identical or identical” (Interviewee, 4) and “You have to reset a bit if you want to find something else” (Interviewee, 1). The starting SFX lacked decent variation, but could be somewhat solved by resetting until you find a desirable starting point. Otherwise, it was generally agreed that it is possible

to progress and find interesting SFX going from the starting point.

All the testers experienced the process to maintain a feeling of progression, that it was generally going in a favorable and expected direction, "I felt there was a progression to speak of" (Interviewee, 2). But 2 out of 4 testers remarked that the process felt slow when strictly trying to find a particular sound effect, "If I had been looking for a very specific sound it would probably take a pretty long time to find" (Interviewee, 3).

3) Tools' utility: Among the tools, evolve was the most used one; one participant explained it as: "Being able to find sounds that will lead me somewhere I want, and being able to identify the parts in the sounds to pass on and evolve" (Interviewee, 3). All the testers had no noticeable difficulties with understanding the effects and purpose of evolving the SFX after trying it out a few times.

The mutation parameters were also a popular tool among the participants. The parameters seem to have helped in affecting the variety, as all testers affirm to have experienced noticing differences when modifying them. This allowed for some amount of control during the process, e.g., one participant used it for removing certain traits from the SFX, "... then I can just mutate that to get exactly that I want ... when it maybe has some weird pitch or something like that" (Interviewee, 1).

The move tool was good for finding other candidate SFX in the current population when the presented set was perceived as poor. 3 out of 4 testers were actively using move during the testing. The other misunderstood how it worked when asked about why.

Save and load were never tested by the participants and play, pause, reset, and retry were rarely used. Only tester 1 used reset and retry when trying to find a particular sound effect. However, when the participants were asked what tools felt least useful, no single tool could be pointed out, "I can't think of anything that is less useful" (Interviewee, 2) and "I could see a purpose with all of them" (Interviewee, 4).

4) Practicality: All agreed that they would possibly use the software given the opportunity for a similar purpose, i.e., for the development of a video game. Two participants commented that they would probably not employ it with the intent of finding a very specific SFX, as noted earlier. Instead, it was suggested that the application is better suited for being surprised or inspired, "Being able to find sounds that I didn't even think I wanted ... it becomes automatically more interesting when it is randomized" (Interviewee, 3) and "You could use it to first be inspired and then try to find something specific" (Interviewee, 4).

5) Suggested improvements: Each participant had different suggestions for improving the artifact:

- Ability to vary the length of the SFX.
- Filter out empty or very similar SFX.
- Support for more audio file formats when exporting, for example, *.ogg*, *.mp3*, etc.
- Ability to select multiple SFX and combine them to one sound when exporting.
- Keep the selection when moving through the population instead of being limited to the presented ones.
- Be able to go back even further using the retry function.
- Stronger colors to highlight which sound effect is currently playing when using the global play button.
- Add an explanation of the bottom status bar to the help section, that which shows the current subset.

V. Analysis

The problem with the expressive range is that it may not completely show an accurate representation of the variety of the content the algorithm is capable of producing. The expressive range gathered in this paper only shows the variety in content from the starting point and does not take into consideration the new sound effects (SFX) that the generator may produce from the users' selection in the longer term.

The participants said to be noticing differences when modifying the mutation parameters and furthermore often preferring to pick a higher value. To support those claims, the expressive range in figure 4 is shown in three configurations where both the mutation parameters are set to 0%, 40%, and 90% respectively. As can be seen in the images, a lower value in mutation results in the sounds being more clustered, while with higher values, they are slightly more dispersed.

The problem with exposing the mutation parameters is that one's experience with evolutionary computation may affect the results. Two of the participants acknowledged that they knew how the parameters functioned from previous experience. For those that did not, it was observed that they still somewhat frequently interacted with that part of the interface, but with less focused intent. This may provide some explanation for the differences among the participants relating to the perception of the generator's quality.

The perception of the generator's quality can also be affected by the random nature of the starting point that the user is presented with. For example, a tester may be presented with a poor set of SFX that makes it difficult to find a laser sound effect, while another may simply be lucky and quickly finds one, and due to one's inexperience in using the software, they may be unaware of how reset can be used to prevent this.

VI. Discussion

Interactive evolution (IE) proved to be a promising approach for generating useful 8-bit sound effects (SFX). The expressive range indicates that the algorithm is capable of producing a variety of SFX with different lengths and volumes. Along with that, the results from the usability tests illustrate that the user can intuitively and efficiently evaluate the broad set of SFX with the help of several assisting tools. The tools provide convenience for the user that likely abates the fatigue and

improves long-term usage. Throughout the process, a feeling of progression is maintained, which indicates that the user experiences that they have some level of control and are able to somewhat predict what the algorithm is going to output based on their selection. The user should not feel like they are a slave to the algorithm, and instead experience that their own creative work comes to fruition. All the participants were fairly interested in the potential of the software and could imagine using it given the opportunity for the development of a video game or something similar, which demonstrates that there is a possible interest for this kind of software in the market.

Another purpose of the software was to be accessible to even those with little to no experience in music or SFX creation. This comes at the cost of lessened control of the process by letting an algorithm do most of the work for you behind the scenes. Depending on the intention behind using the artifact, some may feel the lessened control is preferable, while others may feel uncomfortable with it. Lacking complete control over the process makes it difficult for those with the intention of finding a specific SFX. That is why the software is perhaps better suited for being surprised, inspired, and discovering new SFX, as mentioned by several participants in the interviews. A possible solution that suits both sides is to add the ability for the user to toggle between different levels of control in the process, e.g., a higher level lets the user modify the termination criteria, number of individuals in the population, and more.

An important factor to consider when developing an IE application is providing an intriguing starting point. If it fails to do such, it may result in the inspirational qualities being dulled. For example, the user will find it difficult to find any sound that inspires them and could lead to something interesting. This is especially important when the algorithm is particularly sensitive to producing insubstantial content. The content presented at the start should ideally contain an element of variety, randomness, and richness. However, defining an intriguing starting point comes back around to the reason for why IE was employed in the first place. Instead, as can be seen in *PicBreeder* [8], it is possible to derive the starting point from content that was previously created by other users. The solution in this study strives to somewhat accomplish this idea. By randomly picking multiple entries from a predefined collection of interesting SFX to evolve together, many viable starting points are attained. For example, randomly picking 3 SFX from a collection of 18 results in 816 different combinations. However, this solution has several problems. The first is that the starting point is dependent on the collection, which means that it can be difficult to deviate and find content that lies outside this scope. The other is that the algorithm has a tendency to converge towards a singular model and consequently performs poorly when the provided models are different from each other. This tendency is demonstrated in the results, where some participants notice that many of the SFX in the starting point are fairly similar. Furthermore, the expressive range does display several unusual anomalies that could point to this, i.e., a select few distinct slopes with a higher density. Preferably, the algorithm should find a solid balance between the models and maintain an interesting mix in the population. This study employs several methods to help realize this effect, but it seems they were partially insufficient. For instance, the termination criteria were selected to maintain the diversity before it converges. Incest prevention [16] was implemented as well, but it is unknown if it contributed or not, further tests would have to be conducted where it is compared with and without.

The algorithm's tendency to convergence towards a singular model is most likely due to the fitness not being properly adapt to handling such cases. The problem in detail is presumably related to the evolutionary process and how the fitness function has a preference to give smaller genotypes a higher fitness. Smaller genotypes are probable to have a high similarity to a model because of its lessened complexity, i.e., there are a smaller amount of features to compare, e.g., values in *poke* and *sample*. This issue was discovered during development, when a smaller genotype in the predefined collection always took priority in the population regardless of the combination of models. Attempts were made to fix this as mentioned earlier, but there is still further work needed to be done. There are some possible solutions for this problem that have not been tested: the fitness function could be extended to factor in the genotype's size; the individual is given a higher or lower fitness depending on the number of individuals in the population that are already similar to a particular model. There is also the possibility that the similarity function is not working correctly in certain cases, as its design is experimental with a lack of comprehensive testing.

Among the tools, the mutation parameters seem to be one of the more compelling implementations. The interviews, coupled with the expressive range, show that the mutation parameters do have a noticeable effect on the content. Mutation was specifically chosen to allow the user to vary the variety of content in cases where there is a desire to either view a broader range when there is nothing interesting, or narrow it down when the system is close to the sought sound effect. The expressive range definitely displays that it is possible to achieve this effect in some capacity. Nonetheless, a problem that can stem from exposing these parameters, is possibly causing a misleading perception of the generator's capabilities. The definition of the parameters lies rooted in genetic algorithms, an area most users will probably not have prior experience with. Consequently, inexperienced users may ignore the parameters or set the value to something that, e.g., causes the content to feel like it lacks variation. It is therefore important that a clear and comprehensive description of their exact roles is given to the user.

An emulation of the Commodore 64 SID chip proved as an effective approach for generating 8-bit SFX through the many available parameters. Limiting the sound quality to 8-bit significantly decreases the scope and enables the sound to be commonly interpreted as an old-school sound effect found in many classic games such as *Pac-Man* [25], *Space Invaders* [26],

Asteroids [27], etc. However, the SID is prone to producing sounds that are empty or contain loud dips or pitches. Some of these examples can even be seen in figure 1 and keep in mind that those SFX have a high similarity to the provided models. This signifies that a slight change of value in some parameters can result in these issues. The expressive range does show a slope at the bottom with a rather high density that indicates a lot of silent sounds are being generated. These SFX should preferably just be filtered out. Another way is to expand the fitness function to give a negative score depending on the value of specific parameters in the chip that could be the cause of these issues.

Usually, in interactive evolutionary computation, the population is displayed to the user, they then assess the individuals, another generation passes and repeat. The risk following this approach is possibly requiring numerous evaluations by the user, and thus likely to cause a feeling of a slow and monotonous process. There are several ways of solving this, e.g., decreasing the population size, skipping a few generations, and predicting the user's evaluation. This study aimed to avoid these problems by employing a different approach to allow a much greater population size and more efficient ways of evaluating, while still maintaining a feeling of progression. This approach proved to be promising, but comes with its own set of problems, for instance, performance, defining an accurate similarity function, and difficulties in maintaining the variety.

There are different ways of evaluating the displayed content, e.g., giving a thumbs up or down, directly setting a fitness score, and the Likert scale. To maximize the efficiency of this activity, the chosen method was to directly select the individuals the user prefers while ignoring the rest, as inspired by an identical approach in *PicBreeder*. The activity was simple to understand for the testers and likely improved the overall efficiency regarding the rate of progression.

The methods for reducing user fatigue vary depending on the software and the target audience it is designed for. In this study, the software was primarily designed to be useful for game developers who have limited experience in music or SFX creation and need simple SFX for a video game they are developing. The software is not completely restricted to this target audience, it can also be relevant for video editors and music producers. That said, the software was developed with several quality factors in mind: effectiveness, efficiency, and understandability, which is directly tied to abating user fatigue. A discussion of the quality factors is presented respectively in the following list:

- *Effectiveness* concerns how successful the software is in its intended objective, which is to generate useful 8-bit SFX. This especially relates to the quality of the output from the algorithm, which because of its experimental nature, received a lot of focus during the development stage. The algorithm was contrived with the intention of presenting a set of SFX that visibly derives from the models, but that have been altered to make them slightly different. In this set, the user can find SFX that they may experience to improve upon the models in certain desirable aspects. Otherwise, there is a risk that the user may find it difficult to interpret the outcome from the algorithm and how they are meant to progress. The design of the algorithm was decided to follow the standard genetic algorithm (GA) methodology. There were also thoughts on using a compositional pattern producing network as inspired by *PicBreeder*, *AudioInSpace* [15], and the work by Jónsson et al. [7], but it was deemed too challenging and experimental to be applied and studied within the assigned time period in which to conduct this study. GA was also chosen because it allows for some level of control over the generated content and suitability with the Commodore 64 SID chip, i.e., the genotype representation. The results show that all the participants felt that the SFX that was presented elicited a feeling of progression, which coincides with the goal of the algorithm. There were otherwise a few problems that some of these SFX were very similar to each other, empty or quiet, or contained loud pitches or dips.
- *Efficiency* refers to how quickly the user can achieve their goal using the software. Even if the algorithm is successful in generating useful SFX, the user can still feel discouraged to use the software if the process it accompanies is long and tedious. That is why several of the assisting tools and other features were developed to help make the process overall more efficient for the user, which is: save, load, reset, retry, move, mutation parameters, waveform visualizer, fast performance, simple evaluation activity, and improved starting point. The results demonstrate that several of these features does make a positive impact on the process and is a probable reason for the short durations spent during each iteration of evaluation. For instance, waveform visualizer allows for quick and basic assessment of SFX shown by how the participants did not listen to all the presented SFX, move presents new potentially interesting SFX in the current population which was often used by several of the participants, fast performance reduces dead time, improved starting point presents SFX that the user is likelier to find interesting and progress from, and mutation parameters helped vary the randomness. Save, load, reset, and retry possibly improves efficiency as well but saw limited usage during the usability tests and therefore needs further testing. Efficiency overall needs to be further studied; because the usability tests were conducted in a short-term setting due to not wanting to discourage possible participants, there is a lack of results from the long-term perspective of users employing the software in a practical setting. Such an environment would grant ample data on various aspects of the software that the results in this study have not been able to cover.
- *Understandability* is important because the user should ideally quickly understand how the software works and how it can be interacted with. Otherwise, the user may become overwhelmed or confused over the process that results in premature fatigue. The user interface (UI) was designed to

be simple and minimalistic with only relevant information being presented, e.g., all tools have an associated image to improve recognizability, max 12 SFX are shown at a time to prevent being overwhelmed, and the current progress of the algorithm is displayed when executed for less confusion. It is also important that the user understands how the algorithm and assisting tools works and can be utilized, which was achieved by implementing a small help section that explains such details that the user can access at any time. The results show that after reading the help section, the testers did not have that much difficulty in learning to use the software. The second step in the testing procedure generally only lasted a few minutes before the tester felt satisfied, which shows that it did not take long before they felt they understood the software. The interview also reinforces this: all the testers agreed that they had no significant difficulties when learning to use the software. There were only some minor things that need further work, e.g., explanation of how the separate mutation parameters function, and better highlighting of which sound effect is currently playing. One participant also commented they initially misunderstood how the move tool functioned and therefore did not try it, their proposed solution was to add a small explanation in the help section that explains the bottom status bar in the software which was missing.

Selecting fitting participants for the testing procedure is important to produce valuable data from the observations and interviews. A fit participant is someone who is likely to have an interest in the software and is thus probable to adopt a more critical perspective. The participants in this study were selected based on accessibility and relevancy to one of the target audiences that the software is intended for, i.e., game developers. This selection possibly helped provide more valuable data that perhaps unfit participants would not be able to yield. However, the ideal participant would be someone who is even closer to the target audience; i.e., someone who needs simple SFX for a game they are developing. Another factor that could make the data more valuable is including participants with experience in music or SFX creation. People who have experience in the field are likely to view the generated SFX in a more nuanced way, and are furthermore able to compare between the systems they use and the developed software in this study. This comparison could lead to a different perspective on the practicalities of the software and its effects in the field.

VII. Conclusion & Future Work

This study presented a novel approach for generating 8-bit sound effects (SFX) through interactive evolution and using an emulation of the Commodore 64 SID chip. The user is presented with a large set of various SFX that they can choose to either export or evolve. The evolution optimizes the population towards the users' selection with the goal of providing a rich variation that visibly derives from it. The artifact was developed for the purposes of making SFX creation more accessible, as well as implementing and evaluating several potential methods for reducing user fatigue towards making the software practically viable. An analysis of the expressive range, combined with the results from the interviews, signifies that the software can be utilized intuitively and efficiently for creating new and potentially interesting SFX. The implemented solutions for facilitating a better user experience show considerable promise that can inspire similar works in the domain to adopt or build on.

As for future work, long-term testing on the artifact with more participants would grant a deeper understanding on the practicalities of the software and also the unused tools from the tests. The participant would be given the artifact to use for an extended period of time for practical purposes, e.g., creating SFX for a specific pre-developed video game as the testing environment. In this case, there would be a higher chance that the tester extensively uses the software and its assisting tools and is thus able to form more refined opinions about its usability. They would then be interviewed with similar questions as in figure 3, but that have been expanded and rephrased to present a more nuanced discussion, e.g., "In what capacity did you employ the software?" and "How did you experience the creative process?". The questions could also be expanded to gain additional information about each participant, for instance, experience in music or SFX creation. This would grant a better perspective on the collected answers from the interviews and recognize the differences. Data could also be collected implicitly based on how the tester interacts with the software, e.g., time between each iteration of evaluation, number of times each tool is used, what values the mutation parameters are set to, and how the expressive range changes from the user's selection.

The genotype representation could be expanded with additional commands that allow for more complex SFX to be generated. For example, in Commodore 64, it is possible to declare variables and specify *FOR* or *IF* statements. This would require building a much more complex interpreter to parse these commands correctly.

The collection of SFX used to generate the starting point could be substantially improved. As the collection is predefined and cannot be altered, the user is always stuck within the same scope of diversity, which could decrease the longevity of the software. The collection could be improved by allowing the user to alter it themselves, or to move the collection to a server and regularly add new SFX that originates from previously created content. Whenever a user has found an interesting sound and exports it, the genotype is uploaded and added to the collection if it passes certain criteria, e.g., if it is unique and distinct.

The load functionality has the problem of being restricted to only being capable of loading *.txt* files that adhere to the genotype representation. Modifying the tools to load any kind of short *.wav* file and use it as a starting point would greatly improve the practicality of the artifact. Converting the *.wav* to the equivalent genotype could be done through a search-based

approach. The initialized population would gradually evolve towards the specified *.wav* by comparing the audio samples between the *.wav* and the individual. The similarity could then be calculated and used as a fitness score, where high similarity means greater fitness and vice versa. However, this process would most likely be very slow and susceptible to premature convergence.

From the participants in the usability tests, several suggestions were made that could improve the artifact. For instance, filtering out empty or very similar SFX would make the interactive process significantly more effective for the user by reducing the number of required evaluations. A filter could be implemented that prevents similar or empty individuals from appearing on the UI. Instead, they are replaced with another individual from the population that passes the filter.

REFERENCES

[1] J. Togelius, N. Shaker, and M. Nelson. "Introduction". In: *Procedural Content Generation in Games*. Gewerbestrasse 11, 6330 Cham, Switzerland: Springer International Publishing, 2016, pp. 1–15. DOI: 10.1007/978-3-319-42716-4.

[2] Mojang. *Minecraft*. Mojang and Xbox Game Studios, 2011.

[3] J. Togelius, N. Shaker, and M. Nelson. "The search-based approach". In: *Procedural Content Generation in Games*. Springer, 2016, pp. 17–30.

[4] A. Liapis, G. Smith, and N. Shaker. "Mixed-initiative content creation". In: *Procedural Content Generation in Games*. Springer, 2016, pp. 205–208.

[5] M. Kaliakatsos-Papakostas, M. Epitropakis, A. Floros, and M. Vrahatis. "Controlling interactive evolution of 8-bit melodies with genetic programming". In: *Soft Computing* 16.12 (2012), pp. 1997–2008. ISSN: 1432-7643. DOI: 10.1007/s00500-012-0872-y.

[6] N. Tokui and H. Iba. "Music Composition with Interactive Evolutionary Computation". In: *International Conference on Generative Art* (2000).

[7] B. Jónsson, A. Hooves, and S. Risi. "Interactively Evolving Compositional Sound Synthesis Networks". In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. GECCO '15. Madrid, Spain: Association for Computing Machinery, 2015, pp. 321–328. ISBN: 9781450334723. DOI: 10.1145/2739480.2754796.

[8] J. Secretan, N. Beato, D. D'Ambrosio, A. Rodriguez, A. Campbell, and K. Stanley. "Picbreeder: Evolving Pictures Collaboratively Online". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, 2008, pp. 1759–1768. DOI: 10.1145/1357054.1357328.

[9] K. Peffers, T. Tuunanen, M. Rothenberger, and S. Chatterjee. "A Design Science Research Methodology for Information Systems Research". In: *J. Manage. Inf. Syst.* 24.3 (2007), pp. 45–77. ISSN: 0742-1222. DOI: 10.2753/MIS0742-1222240302.

[10] J. Togelius, E. Kastbjerg, D. Schedl, and G. Yannakakis. "What is Procedural Content Generation? Mario on the Borderline". In: *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*. Association for Computing Machinery, 2011. DOI: 10.1145/2000919.2000922.

[11] H. Takagi. "Interactive evolutionary computation: fusion of the capabilities of EC optimization and human evaluation". In: *Proceedings of the IEEE* 89.9 (2001), pp. 1275–1296. DOI: 10.1109/5.949485.

[12] R. Kamalian, E. Yeh, Y. Zhang, A. Agogino, and H. Takagi. "Reducing Human Fatigue in Interactive Evolutionary Computation Through Fuzzy Systems and Machine Learning Systems". In: *2006 IEEE International Conference on Fuzzy Systems*. 2006, pp. 678–684. ISBN: 0780394887. DOI: 10.1109/FUZZY.2006.1681784.

[13] L. Cardamone, D. Loiacono, and P. Lanzi. "Interactive Evolution for the Procedural Generation of Tracks in a High-End Racing Game". In: *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*. GECCO '11. Dublin, Ireland: Association for Computing Machinery, 2011, pp. 395–402. ISBN: 9781450305570. DOI: 10.1145/2001576.2001631.

[14] E. Hastings, R. Guha, and K. Stanley. "Interactive Evolution of Particle Systems for Computer Graphics and Animation". In: *IEEE Transactions on Evolutionary Computation* 13 (2009), pp. 418–432. DOI: 10.1109/TEVC.2008.2004261.

[15] A. Hoover, W. Cachia, A. Liapis, and Y. Georgios. "AudioInSpace: Exploring the Creative Fusion of Generative Audio, Visuals and Gameplay". In: vol. 9027. Springer International Publishing, 2015, pp. 101–112. ISBN: 978-3-319-16497-7. DOI: 10.1007/978-3-319-16498-4_10.

[16] L. Eshelman and J. Schaffer. "Preventing Premature Convergence in Genetic Algorithms by Preventing Incest". In: *Proceedings of the Fourth International Conference on Genetic Algorithms*. Jan. 1991, pp. 115–122.

[17] N. Shaker, G. Smith, and G. Yannakakis. "Evaluating content generators". In: *Procedural Content Generation in Games*. Springer, 2016, pp. 215–224.

[18] G. Smith and J. Whitehead. "Analyzing the Expressive Range of a Level Generator". In: *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. PCGames '10. Association for Computing Machinery,

2010. ISBN: 9781450300230. DOI: 10.1145/1814256.1814260.

[19] J. Newman. “Driving the SID chip: assembly language, composition and sound design for the C64”. In: *GAME: The Italian Journal of Game Studies* 6 (2017). ISSN: 2280-7705.

[20] *6581 SOUND INTERFACE DEVICE (SID)*. Commodore Semiconductor Group, 1982.

[21] Inc. Commodore Business Machines. “Programming sound and music on your Commodore 64”. In: *Commodore 64 Programmer’s Reference Guide*. Commodore Business Machines, Inc. and Howard W. Sams & Co., Inc., 1982, pp. 183–208.

[22] D. Lem. *reSID*. Version 0.16. June 11, 2004. URL: http://www.zimmers.net/anonftp/pub/cbm/crossplatform/emulators/resid/index.html.

[23] Inc. Commodore Business Machines. “BASIC language vocabulary”. In: *Commodore 64 Programmer’s Reference Guide*. Commodore Business Machines, Inc. and Howard W. Sams & Co., Inc., 1982, pp. 29–97.

[24] J. Baker. “Adaptive Selection Methods for Genetic Algorithms”. In: *Proceedings of the 1st International Conference on Genetic Algorithms*. L. Erlbaum Associates Inc., 1985, pp. 101–106. ISBN: 0805804269.

[25] Namco. *Pac-Man*. Namco and Midway, 1980.

[26] Taito. *Space Invaders*. Taito and Midway, 1978.

[27] Atari, Inc. *Asteroids*. Atari, Inc., 1979.