The University Of Sheffield.

Agent-Based Graphic Sound Synthesis and Acousmatic Composition

Stephen Anthony Pearse

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

Department of Music

The University of Sheffield

October 2016

Abstract

For almost a century composers and engineers have been attempting to create systems that allow drawings and imagery to behave as intuitive and efficient musical scores. Despite the intuitive interactions that these systems afford, they are somewhat underutilised by contemporary composers. The research presented here explores the concept of agency and artificial ecosystems as a means of creating and exploring new graphic sound synthesis algorithms. These algorithms are subsequently designed to investigate the creation of organic musical gesture and texture using granular synthesis. The output of this investigation consists of an original software artefact, *The Agent Tool*, alongside a suite of acousmatic musical works which the former was designed to facilitate. When designing new musical systems for creative exploration with vast parametric controls, careful constraints should be put in place to encourage focused development. In this instance, an evolutionary computing model is utilised as part of an iterative development cycle. Each iteration of the system's development coincides with a composition presented in this portfolio. The features developed as part of this process subsequently serve the author's compositional practice and inspiration. As the software package is designed to be flexible and open ended, each composition represents a refinement of features and controls for the creation of musical gesture and texture. This document subsequently discusses the creative inspirations behind each composition alongside the features and agents that were created. This research is contextualised through a review of established literature on graphic sound synthesis, evolutionary musical computing and ecosystemic approaches to sound synthesis and control.

Acknowledgements

This work is dedicated to the amazing people that have supported me over the course of this research; I will be forever grateful for their support, guidance and patience. In no specific order I would like to thank:

My parents Graham and Sandra and Grandmother Joyce for their unwavering belief and support over these difficult years. My younger sister Lydia for continuously challenging and supporting me. My supervisor, Dr Adrian Moore for his guidance, ears and time over the years. My good friend Dr Dave Moore for his inspiration, advice and above all, patience. Martin Curtis-Powell for his support and friendship along with his eye for mistakes. Professor Leigh Landy for his encouragement and support. I would also like to thank my close friends who have helped me over the course of this journey, Alexander Farr, David Devaney, James Peruzzo, James Surgenor, Jordan West, Ruben Tojeiro-Gonzalez and Tom West. I would also like to give special thanks to the D'Arcy family along with Samantha and Ruby-Rose Eastwood who have all provided so much joy and encouragement over this period. I would also like to dedicate it to my chameleon Frank and childhood cat Buzz, who sadly passed shortly before this submission.

Table of Contents

Table of Figures

Chapter 1. Introduction

The research presented here contains a portfolio of fixed medium acousmatic compositions alongside a reprogrammable agent-based control environment that was designed to generate materials for these works. The compositions adhere to the definition of acousmatic music proposed by Dhomont (1996) in that they are specifically intended for presentation over loudspeakers. As part of the composition process, a complex and reprogrammable agent-based graphic sound synthesis system was created to enable the exploration of ecosystemic granular synthesis algorithms as a means of creating evolving textures and gestures. These original synthesis algorithms are designed to enable the exploration of new forms of image sonification and audification in an open and flexible manner. Each composition presented here subsequently used imagery as a means of stimulating granular synthesis algorithms in real-time. The software output *The Agent Tool* (Pearse and Moore, 2014) was developed using an iterative methodology based on the principles of *agile* development (Paetsch et al., 2003). Each iteration of the software was developed as a means of facilitating the compositional aims of each musical composition presented here. After completing each of the five musical works, the software feature-set and methodology used was subsequently evaluated before the commencement of the following composition and iteration of the software. Across these works, the system was actively used as a means of generating sonic materials that form *part* of the compositional process rather than entire compositions. While the software allows entire compositions to be crafted within it, akin to Waschka II (2007), the author believes that these algorithmic techniques should serve the user rather than the other way around.

Chapter 2 discusses the compositional rationale adopted across the works presented alongside subsequent research aims. Aims relating to the software artefact are subsequently explored and justified through a review of contemporary literature in Chapter 3. Over the course

of this chapter literature on graphic sound synthesis, evolutionary computing algorithms and ecosystemic agent systems in musical composition will be discussed. Chapters 4-8 discuss each composition and subsequent iteration of The Agent Tool. Within each of these chapters the aims and intentions of the respective composition are examined before presenting how The Agent Tool was developed and utilised to facilitate the compositional aims. Example agents and timings are subsequently presented. Each of these chapters concludes with an evaluation of the agent methodologies utilised within that work. Chapter 9 concludes the document evaluating the compositions and software artefact against the previously discussed research aims.

This thesis also presents a list of public performances of presented compositions as an appendix. An accompanying DVD is provided with all musical works and installers for the software artefact. The DVD (and installer) also contains a comprehensive API reference manual and getting started guide that could not be included here due to their size.

Chapter 2. Research Aims

The compositions presented here aim to explore the development of organic musical textures and gestures. Texture as put forward by Smalley (1986) is typified by materials with internal energy and motion that do not require provocation to act. The materials subsequently appear to develop and evolve without human interaction. Gesture on the other hand is concerned with "action directed away from a previous goal or towards a new goal; it is concerned with the application of energy and its consequences" (Smalley, 1986, p.82). Gesture, unlike texture, is typified through energy that is built up, stored as tension and released externally. Gestures require provocation and purpose within a work. These materials are often derived or emerge from existing materials that are present in the work and are used to drive the composition forward. As Smalley (1997) suggested, across both of these functions the development of materials can be comprised of three components. Materials are presented to the listener in the form of onsets before they continue (continuants) in some form before termination. This research aims to explore an organic and somewhat evolutionary approach to the development of these morphologies within composition. The Agent Tool, as presented here was subsequently designed to facilitate these explorations using imagery and agency as stimuli for these endeavours.

Composers have been using forms of graphic sound synthesis and image sonification as a means of realising their compositional ideas for some time. Typically these systems use additive synthesis to create synthetic materials directly from the colour data within an image. This approach gives rise to materials that are highly synthetic in their output. Granular synthesis algorithms, as put forward by Roads (2001), are constructed through the sequencing of grains of microsound (<50ms in duration). These grains are most often sourced from one or more pre-recorded sound files or buffers and can offer greater diversity in their sonic output compared to additive synthesis. To subsequently generate sonic materials via granular synthesis, a user needs to

spend time exploring the contents of the sound files being used to comprehend the sonic possibilities that they afford. The parametric space and favourable settings within a granular process subsequently change with every sound-file that is used. Over the past two decades, artists and composers have explored methodologies for optimising exploration within the parameter space of granular synthesis algorithms. Evolutionary algorithms have successfully been utilised to do this (see Dahlstedt, 2001, Mandelis, 2001) but have also been used to generate synthesis algorithms themselves (see Garcia 2001). Evolving Artificial Life (a-life) systems have also been utilised as a means of generating and controlling granular synthesis algorithms in an ecosystemic fashion (see Miranda, 2000, Eldridge, 2005, McCormack, 2001).

The research presented here questions whether agent-based/a-life graphic sound synthesis is an effective means of creating sonic gesture and texture via granular synthesis. More broadly, The Agent Tool aims to investigate how agency can be used to explore open-ended parametric spaces. As part of this, the research aims to explore whether agency can be used as an effective means of designing both adaptable and intuitive instruments/controllers to perform compositional tasks. Finally, an investigation of imagery as an effective modality for invoking and stimulating sonic ecosystems when designing textures and gestures is discussed.

Chapter 3. Existing Research

Section 3.01 Graphic Sound Synthesis and Image Sonification

For almost a century composers and engineers have been attempting to utilise imagery as a means of creating compositional materials and methodologies. As Xenakis famously noted, "Why should I write them as (musical) notes instead of these lines? We are used to seeing things in visual shapes; it is natural" (in Chadabe, 1997, p.213). The collection of hardware and software systems that have been developed for this purpose perform what is known as Graphic Sound Synthesis (Roads, 1996). These systems subsequently perform forms of image sonification and audification. Sonification is typically defined as a system in which numerical relationships *within* data are realised in sound that is to be perceived by a listener (De Campo, 2007, Hermann, 2008, Kramer, 1993, Scaletti, 1994, Walker and Nees, 2011). Audification is a form of *direct* sonification wherein the data *itself* is realised in sound (De Campo, 2007, Kramer, 1993, Walker and Nees, 2011). These two approaches to sound and music creation can be viewed as opposing ends of a continuum. As put forward by Kramer (1993), sonification is a *symbolic* realisation of the data which is abstracted and interpreted. Audification, on the other hand, is *analogical* as it is a direct and intrinsic representation of the data. The terms *symbolic* and *analogical* often vary depending on the target audience for the realisation. Hogg and Vickers noted, when these realisations are used to create art in the *ars musica* domain, the terms *abstract* and *concrete* are often used over *symbolic* and *analogical* (Hogg and Vickers, 2006).

Within the field of sonification, De Campo categorises three broad approaches to realising data (De Campo, 2007).

1. Event-based
2. Model-based
3. Continuous

Graphic Sound Synthesis systems typically utilise forms of continuous realisation to generate sound. De Campo proposed that the rate of data extraction directly corresponds to whether a realisation is symbolic or analogical, consequently, a sonification over audification (De Campo, 2007). Extracting colour data at high rates, from this perspective, can be viewed as an audification. Traditionally a high rate of extraction implies that the image data should be used as raw sound. However, it is also possible to use this data as control data. Traditionally this would take the form of an off-line process, but huge increases in CPU speed and access to the parallel processing potential of commonplace GPUs have allowed these high intensive process to be calculated in real-time[1]. In the realm of Graphic Sound Synthesis, the vast majority of systems adhere to performing direct image audification. These include L. Lavellèe's *sonothèque*, Hugh LeCaine's *Coded Music Apparatus*, the *Cross-Grainger Free Music Machine,* O.Kendall's *Composer-Tron,* E.Murzin's *ANS Synthesiser* along with contemporary software systems *MetaSynth* and *PhotoSounder* (Rhea, 1972, Roads, 1996, Rouzic, 2008, Wenger, 2015, Young, 1989). These systems are designed to read an image's content from left to right. The vertical axis is then split into a discrete collection of bands, each representing a frequency or collection of frequencies. Each of these has a corresponding oscillator (typically a sinusoidal wave) whose amplitude is controlled by the brightness of the current pixel. Thiebaut et al. (2008) suggested that in the process of scoring musical materials graphically, composers leave significant amounts of vagueness in their scoring. Due to the somewhat empirical nature of audification, contemporary systems that utilise this methodology tend to be used explicitly for sound design (on a sound by sound basis) and not composition (Thiebaut et al., 2008).

Parameter Mapping Sonification (PMSon) is another favourable methodology. Data, or attributes within, are mapped to acoustic attributes such as pitch, timbre, spatial panning and so

[1] The real-time single sample phase vocoder by Bradford and Dobson accurately highlights this potential (Bradford, R &Dobson, R. 2011).

on (Hermann and Hunt, 2005, Ben-Tal et al., 2002, Hermann and Hunt, 2004, Kramer et al., 2010). This approach is particularly well suited for sonifying multivariate data sources as sound is inherently multi-dimensional. Consequently, numerous authors have highlighted this approach being the most favourable sonification strategy amongst creative practitioners and composers (Flowers, 2005, Flowers et al., 1997, Grond and Berger, 2011, Hermann, 2008, Hermann and Hunt, 2004, Hermann and Hunt, 2005, Kramer et al., 2010, Walker and Nees, 2011, Worrall, 2010). Two notable Graphic Sound Synthesis systems adhere to this methodology, Daphne Oram's *Oramics Machine* along with Iannis Xenakis' Unitè Polyagogique Informatique de CEMAMu (UPIC) system (Chadabe, 1997, Lohner, 1986, Manning, 2012, Manning, 2013, Marino et al., 1993, Roads, 1996). Walker and Nees (2011) suggest that PMSon typically forces a passive mode of interaction with a system once it is running. It has been noted that both the Oramics Machine and the various iterations of UPIC, were designed to allow users to alter parameter mappings through voltage control and the *arc* system within UPIC (Lohner, 1986, Manning, 2012, Marino et al., 1993). As indicated by Chadabe, this flexibility in UPIC resulted in it being used by a wide variety of composers[2] throughout the late 1980s and early 1990s (Chadabe, 1997). The real-time software version of the UPIC (released in 1988) allowed a user to draw two-dimensional arcs that could control a multitude of parameters. This included the speed of playback of each of the sixty-four voices. Arcs were stored in *pages* which a user could traverse, with each page supporting playback of 4000 arcs at once (Manning, 2013). The system, therefore, offered an enormous number of parametric combinations for a user to explore. Magnusson indicates that being faced with a nearly infinite expressive scope can result in a creative paralysis (Magnusson, 2010). The UPIC system was successful as it allowed users to constrain effectively their explorations enabling more focused interactions.

[2] Notable composers include Bernard Parmegiani, Jean-Claude Eloy, François-Bernard Mâche, Julio Estrada, Cort Lippe, Jean-Claude Risset, Joji Yuasa and Curtis Roads (Chadabe, 1997).

Unlike, scanning/audification systems, Xenakis made it clear that UPIC was not designed for the sonification of images. For Xenakis, compositions rather than imagery should remain at the heart of the activity (Marino et al., 1993). Within this context, imagery merely provided an efficient means of scoring highly complex sonic materials. If a flexible image sonification system were to be designed with the parametric freedom of UPIC while allowing colour data to be freely mapped, careful constraints and control mechanisms for control must be offered to the user.

The work of Yeo and Berger has explored two concepts of time mapping in image sonification (Yeo et al., 2004, Yeo and Berger, 2006, Yeo and Berger, 2005b, Yeo and Berger, 2005a). UPIC allowed individual voices to have independent speeds; most other graphic sound synthesis systems stipulated that all voices advance through time in a unified fashion. Yeo and Berger proposed what they call *probing* (Yeo and Berger, 2005b). This technique allows a user to select points within the image to sonify and then have them navigate the space freely as independent entities. This approach to free traversal of an image was also put forward by Sedes et al. (2004) wherein images were utilised as two-dimensional terrains over greyscale images. Free traversal within an image can be viewed as a model-based sonification requiring exploration by human or automated entities.

Kramer et al. (2010) propose that sonification system development, when used to aid sonic exploration should reference the following:

1. Control: systems should provide efficient, effective and parametric controls
2. Mapping: should allow the design of new sonifications 'on the fly' by giving the user flexible, intuitive control over mappings
3. Integrability: facilitate data importation from a variety of formats
4. Synchrony: allow easy integration with other systems such as visual monitors

5. Experimentation: integrate a perceptual testing framework with the overall sound synthesis and mapping functions.

For experimentations and exploration in new forms of graphic sound synthesis, these key points should be adhered to. Due to the multidimensional nature of the data set and a plethora of potential synthesis mappings, careful experimentation of new mapping strategies should be explored.

Section 3.02 Evolutionary Algorithms in Composition and Instrument Design

Since the pioneering work of John Holland (1975) and Richard Dawkins' *BioMorph* framework (Dawkins, 1986), Evolutionary Algorithms (EA) systems and Genetic Algorithms (GA) have been used as a means of fulfilling a variety of musical tasks by researchers. These algorithms have been used to create musical compositions that suit the style of a composer. As Husbands et al. (2007) along with Burton and Vladimirova (1999) have indicated, many of these have been highly successful (Biles, 1994, Biles, 1998, Biles and Eign, 1995, Bilotta et al., 2002, Burton and Vladimirova, 1999, Gartland-Jones and Copley, 2003, Hodgson, 1999, Hodgson, 2002, Jacob, 1995). The *GenJam* system devised by Biles (Biles, 1994, Biles, 1998, Biles, 2007, Biles and Eign, 1995) is acknowledged as being one of the most sophisticated implementations and has undergone several successful iterations (Husbands et al., 2007, Johnson, 1999). GenJam is designed to generate jazz solos whose output and *fitness function* can be influenced by a user and audience members (Biles and Eign, 1995). The multiple compositions of Waschka II (2007) with the aid of *GenDash* have evidenced that GA can be a highly effective means of generating both components for composition and compositions themselves. As demonstrated by Garcia (2001) and Johnson (1999), both EA and GA have also been utilised to explore the creation of new synthesis algorithms.

In the realm of parametric design and control, Evolutionary Algorithms have proved to be a highly efficient and engaging methodology. As Dahlstedt (2001a) noted:

> *A multi-dimensional parameter space, such as the parameter set of a synthesis algorithm, is an ideal target for interactive evolution. The individuals can be parameter sets for sound engines or pattern generators, like a synthesiser preset consisting of 100 parameters or more. You can navigate the huge parameter space without any knowledge of its underlying structure, following your esthetical preferences, and still have a very high degree of control* (Dahlstedt, 2001a, p.2).

Upon discussing the four roles in which Evolutionary Algorithms can be utilised within a work (sound design, material generation, structure generation and large-scale form generation) Dahlstedt (2001a) suggests that these methods work best at a low level for material and structure generation. Furthermore, the way in which materials are often utilised determines the quality of the work rather than the materials themselves, due to listener's expectation of coherence and consistency (Dahlstedt, 2001c). Perhaps most importantly, he noted that when generating such materials within a large parametric space, predicting the impact of parametric manipulations can be problematic. Interactive Evolution can, in turn, be utilised to maximise the proportion of good results (Dahlstedt, 2001c). Historically EA and GA have been utilised as a generalised tool for optimisation (Dahlstedt, 2009, Horner and Goldberg, 1991, Husbands et al., 2007). These processes are based upon Darwin's theory of natural selection (Darwin, 1897) in which results are iteratively refined until a preferred output is discovered. This process can be summarised as follows:

1. A random starting population is generated
2. A user or fitness function tests the sounds and selects their favourites
3. A new population is generated based upon variations of the previously selected population
4. The process is repeated from step 2

As part of this process, the objects or mappings are stored within a genetic representation known as the genome. This can be a string of text or numbers, a tree structure or programming code. A single instance of a genome (called a genotype) is assessed by the user or a fitness function for suitability. This suitability is measured on a numerical scale before the genome is altered by some means to produce a phenome with relevant phenotypes (variations) within. As Husbands et al. (2007) indicated, if automatic suitability testing takes place, this fitness function must somehow codify the desired musical outcome. When such systems are being used to explore an unknown parametric space creatively, codifying an effective fitness function is problematic. As Biles and Eign (1995) state, in subjective domains, fitness functions often cannot be expressed algorithmically. Similarly, as Husbands et al. (2007) noted, when EA are being used to develop new sounds, unlike composition systems there may not be an equivalent source to assess the 'quality' of the sound. Furthermore, due to the complexity of audio synthesis systems and the difficulty in modelling aesthetic judgements, Husbands et al. (2007) justly state that in this domain the subjective rules over the objective. Subsequently, a common but time-consuming solution to this problem is to use human judgement as a fitness measure (Husbands et al., 2007, Sims, 1991, Todd and Latham, 1992, Burton and Vladimirova, 1999).

The use of human judgement as a fitness function has been prevalent amongst researchers using EA to compose higher level musical structures. Across the various iteration of Bile's influential GenJam system, human judgment is used to create a narrow *fitness bottleneck*.

> *…what if no suitable algorithm exists for generating fitness, even in the minimal case of deciding which of two individuals is better? After initially considering a neural network trained to respond as I do to pieces of music, I decided to put off the search for an algorithm that implements "I know what I like," and use myself as a fitness function… The human who serves as GenJam's mentor, then, is a very narrow bottleneck and is, in fact, the limiting factor on population sizes, number of generations, and size of any generation gap. This is because the mentor must*

> *listen carefully to every measure of every phrase to provide their fitnesses* (Biles, 1994, p.135).

GenJam was subsequently developed into an Interactive Genetic Algorithm (IGA) to facilitate both *audience-mediated performance* situations[3] and real-time interactive performance based upon live performer input (Biles, 1998, Biles and Eign, 1995). The *Variations* system of Bruce Jacob (1995) and *IndagoSonus* by Gartland-Jones and Copley (2003) take a higher level approach to music generation. Both of these frameworks use higher level abstractions to represent musical phrases. These *blocks* are subsequently developed and given the ability to be influenced by phrases that are passed to them. In the case of the Agent Tool which is presented here, this approach is adhered to in the form of motion models (see Section 5.01(f)) which can be subsequently influenced by agents or the user.

In the realm of developing new sounds through parametric exploration, similar bottleneck techniques are used with some variations on the iteration process (Dahlstedt, 2001c, Husbands et al., 2007, Mandelis, 2001, Mandelis and Husbands, 2003). Whilst this is predominantly the case, when EA systems have been used to model particular sounds accurately (Garcia, 2001), target sound files have used with an algorithmic fitness function that compares characteristics of the resultant audio files with the target. This process is a convergent one where the synthesis algorithms and parametric controls converge on a target state. When these systems are used to explore new sound creation with a human bottleneck, parametric evolutions often diverge from the source genotype (see Mandelis, 2001) as a means of presenting new parametric possibilities. As Husbands et al. (2007) put forward, these overarching processes vary with one being convergent and the other divergent, which they liken to 'survival of the fittest' and 'survival of the prettiest'.

[3] In this iteration members of an audience could preview variations and act as the mentor/bottleneck prior to a performance (Biles and Eign, 1995).

The *MutaSynth* and *Genophone* systems by Dahlstedt and Mandelis respectively are designed specifically to traverse a parameter space as a means of creating new sounds and control mechanisms (Dahlstedt, 2001a, Dahlstedt, 2001c, Mandelis, 2001, Mandelis, 2002). MutaSynth is split into two distinct components, the genetic modification programme and a separate sound engine which communicate through MIDI Continuous Controller (CC) and System Exclusive Messages (Sysex). The framework stores a parameter mapping genome as a string of numbers. The collection of genetic operators can consequently be used to generate a collection of nine new phenomes which a user can quickly audition using a numerical keypad (Dahlstedt, 2001c). Across the genetic operators that the system affords, a user can alter settings of each. For example, the *mutation* operator allows a user to set the probability of random alteration and the maximum mutation range. The *mating* (crossover) and *insemination* (asymmetrical crossover) operators allow two parent genomes to be combined, controlling a genotype's inheritence in the child. Manual mutation is also possible within the system. Upon discussing the system, Dahlstedt noted that when dealing with large sound engines, parameters should be divided into groups with mutations being applied to each of these individually if required to limit the number of bad mutations (Dahlstedt, 2001b). Perhaps more significantly he noted that the system should begin the process of evolution from genomes that are already somewhat successful (Dahlstedt, 2001a, Dahlstedt, 2001c, Dahlstedt, 2007). However, for this to be achieved a user needs to have an awareness of the genome material so as not to disregard the genomes that they deem to be successful. This approach is echoed in the research by Mandelis in the design and usage of the hyperinstrument, the Genophone (Mandelis, 2002, Mandelis, 2001, Mandelis and Husbands, 2003). Husbands et al. (2007) take this further by suggesting that it is not desirable for the Genophone to start with random genomes when exploring new sound.

> *These experiments indicated that it is preferable to seed the initial population with sounds that have been professionally hand-designed and are of some*

> *aesthetic quality. A large amount of knowledge is embedded in the parametric definitions of these sounds, information that ultimately encodes a set of aesthetic values, albeit in an implicit and not easily decipherable way. By using such hand-designed sounds as points of departure for the evolutionary search, this embedded knowledge can be exploited* (Husbands et al., 2007, p.18).

In a similar fashion to MutaSynth, the Genophone allows a user to control arbitrary synthesiser controls using a data glove via MIDI CC and Sysex commands. Genotypes within the Genophone encode both parameter mappings and the current parametric values within the parametric space (Husbands et al., 2007, Mandelis, 2002). The system also allows a user to protect (lock) specific genotypes as part of the evolutionary process as a means of guiding the exploration process.

Genetic representation of the data plays a crucial role in the creative possibilities that the system affords in establishing the space for possible results and enabling modification. (Dahlstedt, 2007). While representations should be as generic as possible to allow for surprises and inspiration beyond an artist's expectation, they should be sufficiently meaningful to allow efficient convergence upon ideas. Dahlstedt (2007) indicates that these representations can be split into three forms, *basic, object-based* and *generative.* Each of these three representations can be summarised as follows:

Table 3.1 Genetic Representations in imagery, based on Dahlstedt (2007)

Genetic Representation	**Example Image Genome Format**	**Subject of Transformation**
Basic	Raw image data containing unfiltered materials	Raw pixel data
Object-Based	Vector graphics containing a list of graphical objects along with information about each	Properties of item/object
Generative	Structural information between items within the data	Relative relationships such as position and size differentials

While Dahlstedt (2007) uses the evolutionary transformation of imagery to describe these representations, these strata can also be used to summarise forms of image sonification. Analogical and symbolic realisations take the form of basic and object-based representations. The arc system of Xenakis' UPIC can subsequently be likened to an object-based representation with each arc being an effective vector within a space. Artificial Life systems (a-life), as we shall see, have been used to perform a generative traversal of parameters and perform generic musical tasks. The emergent and somewhat organic behaviours that form a-life systems are appealing as a means of generating both new forms of parametric control and new sonification algorithms.

Section 3.03 Ecosystemic Approaches to Composition and Sound Design

For many years biological ecosystems at a variety of metaphorical levels have been used as a means of creative discovery (McCormack, 2012). In such systems, rather than creating complex top-down systems for control or creation, individual entities within an ecosystem have the capacity to make decisions and interact with others to create what Eigenfeldt (2010) states as successful complex, dynamic and emergent systems. These systems model a form of a-life to create new methods and paradigms for studying, producing, managing and creating music (Bilotta et al., 2002). The potential for these systems as creative tools has been suggested by numerous authors (Burtner, 2006, Dahlstedt and Mcburney, 2006, Eigenfeldt, 2010, Eigenfeldt and Pasquier, 2011, Murray-Rust, 2008, Murray-Rust et al., 2006, Spicer, 2004, Wooldridge and Jennings, 1995, Wulfhorst et al., 2003, Whalley, 2009). Importantly, as Martins and Miranda (2008) state, a-life systems have the potential to avoid some of the pitfalls of EA as they are not designed to generate efficient solutions to problems automatically. A-Life systems are underpinned by combinations of entities known as *agents*.

Woolridge (2002, p.15) states broadly that "an agent is a computer system, situated in some environment, that is capable of autonomous action in this environment in order to meet its design objectives". However, as Eigenfeldt (2008) noted, the definition and concept of agency has varied greatly across authors and researchers. The original concept of agency within a creative context, as proposed by Minsky (1988) is based on very simple abstractions that subsequently require considerable user actions to yield complex results (Eigenfeldt, 2008). The concept of agency in a-life systems has been used in a variety of industrial and military applications in areas such as scheduling, simulation, architecture, engineering, construction and business (Whalley, 2009). Across all of these, an agent is an entity that acts on behalf of another within a process (Whalley, 2009).

The usage of agents can be split into three categories. The most commonplace system is that of the *Multi-Agent System* (MAS) in which agents compete or work with one another to achieve an individual or collective task. The second approach is that of *Distributed Artificial Intelligence* (DAI) in which agents are utilised to solve a problem or task. The final category is that of a *Multi-Agent Based Simulation* (MABS) which model complex situations and systems. DAI systems have been used extensively in the field of machine learning. Projects such as OMax (Assayag, 2014), the work by Gimenes et al. (2006) along with Goto and Muraoka (1996), Dixon (2000) and Goto (2001) have successfully used agency as a means of identifying musical structures, stylistic traits of performers and perform low-level beat tracking. The *Ensemble* framework by Bruel and Queiroz (2014) can somewhat be considered as a *Multi-Agent Based Simulation* as agents are designed to listen and respond to a collection of internal and external audio sources to form part of a simulated space.

Of the three categories, it is the *Multi-Agent System* which is most appealing to the creative practitioner. As Eigenfeldt (2008), Burtner (2006), Dahlstedt and Mcburney (2006) have all stated,

multi-agent interactive systems offer the possibility for new complex behaviours in systems that can yield organic structures most similar to ecological systems. Whilst many MAS based approaches have been used to perform machine learning algorithms (Goto and Muraoka, 1996, Dixon, 2000) they have also been used as sources of interactive music composition in a variety of different forms. (Miranda, 2000, Miranda, 2001b, Miranda, 2003, Murray-Rust et al., 2006, Murray-Rust and Smaill, 2005, Murray-Rust, 2008, Wulfhorst et al., 2003, Wulfhorst et al., 2001, Spicer, 2004, Dahlstedt and Mcburney, 2006, Eigenfeldt, 2010, Eigenfeldt, 2007, Eigenfeldt and Pasquier, 2011). Agency has been used to generate music and control data from the microscopic level (Miranda, 2000) to higher level meso/macro musical phrases and structures (Eldridge, 2005). One criticism of broader agent-based systems put forward by Paine (2002, p.297) is that "human agent(s) *interact*, but the machine largely *reacts"*. While this is the case with the aforementioned DAIs and MABs, it is not entirely valid when aimed at MAS systems. These systems are primarily *proactive* in creating new means of exploration. As Eigenfeldt (2008) highlighted, the system proposed by Burtner (2006) is reactionary, and an anomaly when compared to other MAS systems.

Many practitioners have used cellular automata as a MAS system across varying timescales. The *Chemical Oscillator* (ChaOS) and *CAMUS* by Miranda utilise the Game of Life with cells/agents within the systems creating grains of microsound or musical notes respectively (Miranda, 2001a, Miranda, 1995, Miranda, 2000, Miranda, 2001b, Miranda, 2003). The *LiveCell* framework by Ash and Stavropoulos (2011) utilises the latter approach but is designed for live performance and generates materials in real-time for a string quartet. The *Eden* project by McCormack populates an artificial world with agents which can move around, monitor and generate new musical materials (McCormack, 2012, McCormack, 2001, McCormack, 2003). This framework actively utilises stigmergy whereby agents can influence the actions of others through mediating constructs (such as virtual food sources) which agents are attracted to. Furthermore, agents within Eden evolve over time to create what Husbands et al. (2007, p.9) deem to be a "highly engaging interactive

artwork". Genealogy and EA have similarly been applied by Malt (2004) in the *Khorwa* framework. Here, each interaction results in an agent redefining its genetic structure before generating offspring that exhibit such traits (Whalley, 2009).

A-life systems are inherently appealing as a means of generating micro and mesoscopic materials and it is fairly common to see agents mapped to individual sound objects. Several important systems have used agency and ecosystems as a means of generating higher level structural materials. *Coming Together* by Eigenfeldt (2010) utilises agency as a means of generating musical phrases that are musically meaningful to one another using appropriate harmonic, melodic and rhythmic associations. Agents within the system subsequently monitor the output of one another and react as a means of moving towards a collective goal. AALIVE and AALIVENET by Spicer uses agency in a similar fashion but instead generates relevant phrases and structures based upon live musical input and stimuli (Spicer, 2004, Spicer et al., 2003). This approach to using agency has parallels with the *VirtuaLatin* system by Murray-Rust et al. (2005) which utilises agency as a methodology for generating Latin-based rhythmical patterns and structures with variations and fills.

As both Dartnall (2002) and McCormack (2012) identify, the concept of 'combinationalism' is somewhat problematic in the field of computational creativity. Combinationalism is the theory that "creativity is the creative combination or re-combination of previously existing elements" (Dartnall, 2002, p.14). *Combinatoric emergence* is the practice of combining existing primitives to generate new outputs. As McCormack (2012) notes, the output of the system is consequently dependent on the knowledge embedded in the base primitives that are subsequently combined. *Creative emergence* as first put forward by Cariani (1991), is proposed as a means in which the process actively alters and iteratively refines the base primitives that are being used to generate the creative output (Cariani, 1991, Cariani, 1997, McCormack, 2012, Eldridge, 2005). This ability to

refine or re-implement the primitives within an a-life system, either through human interference or an automated process is highly significant. Agents and their internal logic can subsequently be viewed as these base primitives. If agents can be altered, they will exhibit one of Eldridges' three desirable characteristics within an a-life system in providing a user "inexhaustible and extremely variable outcomes" (Eldridge, 2005, p.3). The importance of flexibility in design is echoed by Murray-Rust et al. (2005) who draw direct parallels between a system of agents and a collection of improvising musicians. Firstly they must exhibit *situatedness*, an awareness of their environment and surroundings. They must act with *autonomy* and not simply wait for stimulus. Finally they must offer *flexibility* if they are to be used successfully, currently and in the future.

Many implementations mentioned thus far exhibit situatedness and autonomy; a few are also flexible in that they can be repurposed for other musical works and roles. Systems such as Andante (Ueda and Kon, 2004), AALIVENET (Spicer, 2004), Ensemble (Bruel and Queiroz, 2014), Musings (Spicer, 2014) are designed to allow re-programmability in either Java, Pure Data and Max/MSP respectively but users require extensive knowledge of both the language and the custom libraries that have been crafted. For maximum creative emergence from an agent based system, this overhead should be reduced to enable more efficient explorations. If an abstract approach (after Minsky (1988)) is adopted, there is the potential for systems to use a multitude of approaches across varying timescales using a variety of modalities, such as imagery.

Section 3.04 Summary of Methodology

For an a-life framework to be successful in promoting new explorations of graphic sound synthesis, it should adhere to Murray-Rust et al. (2005) and promote situatedness, integrability and most importantly flexibility. Evolutionary Algorithms maximise the potential for creative exploration and emergence and enable the usage of agents and their ecosystems. The Agent Tool uses strings of text containing code as genetic representation. Within this form, an open Application

Programming Interface (API) can result in the code taking the form of an object or generative representation. The development of the agents and ecosystems presented here utilise an adapted form of the EA development cycle. In this process, agents inherit a set of open and flexible mechanisms for future use. These agents are subsequently explored through real-time invocation and control. As these agents are designed for creative exploration, a human fitness bottleneck is used to evaluate the control, performance and sonic output both as an individual and within an ecosystem. The logic and controls that influence the agent's traversal through the parametric space has then been altered before being tested again.

Chapter 4. Voice Based Graphic Sound Synthesis: *Overture* (2012) – 5:38

Overture is a stereo composition that was completed at the University of Sheffield Sound Studios in 2012. It received its premiere at the *International Festival for Innovations in Music Production and Composition* in 2013. It was subsequently performed at the New York City Electroacoustic Music Festival. At the time of the work's conception, *Overture* was designed to be the opening movement of a large-scale acousmatic work dedicated to exploring the spectral purity of the female voice. As such the work is experimental and introductory. While the parent work was never completed, the compositional ideas and materials generated throughout this period made their way into the composition *Liten Röst* (see Chapter 7). This work is a spiritual precursor to all of the compositional research that follows. Compositions completed before *Overture* (see the accompanying DVD) were mainly through-composed with the structure of the work emerging through spectral similarity and contrast in a form of abstracted syntax (Emmerson, 1986b). Spaces and materials in these previous works would often emerge in very quick succession with a great deal of aggression and variety. The first Agent Tool prototype was completed as part of a composition process facilitating a more thematic and texture-carried approach to composition. The system allowed for the real-time exploration of parameter spaces controlling granular synthesis algorithms. The majority of the materials that form the basis of *Overture* were generated through real-time performance with the prototype system.

The musical materials and the structure of the work were derived from real-time exploration of tonal musical material by three distinct voices within a parameter space. The first prototype of the Agent Tool at this time was known as the Draw Tool, and it allowed a user to make use of images as a means of adding extra dimensions to this parameter space. Throughout the development of the composition and software, a collection of images comprising subtle colour

gradients were created as a means of generating continuously evolving parameters (see Section 4.01). In the context of *Overture,* three voices with identical mappings were used. The parametric automation dictated by the colour attributes for each voice consequently became far more noticeable when working with the system and assisted the design of harmonies throughout the piece. The work is inspired in part by the harmonic tendencies found in the dronescapes of the works of *Five Panels* by Matthew Adkins (2009) and the usage of underlying harmonic content in *Forêt Profonde* by Francis Dhomont (1996). Where the harmonic content of *Five Panels* is primarily static over the course of each movement, the underlying harmonic content in *Overture* deliberately changes throughout creating continuous tension and release.

The prototype Agent Tool assisted the creation of harmonic relationships deliberately imposed upon the sonic materials in question. The opening drone presented clear harmonic partials which act as a tonal centre. Contrasting materials evolve and supplant this drone throughout the piece. Once materials had been generated, contrasting harmonic intervals were imposed or implied through careful filtering and convolution, while attempting to preserve internal energy and motion. This can be heard most prominently in the descending textures between 03:17-03:34. The emergence of these textural materials at 02:30 marks the end of section *A* before moving to more agitated textural materials in *B*. Further examples of this are as follows.

Table 4.1 Example harmonic transformations in Overture

Time of introduction	**Description**
1:35	Introduction of a clear harmonic drone
1:50	Gestural elements – harmonic drone now acts as a pedal in contrast to grain-based materials
2:06	Climactic gesture with continued harmonic underpinning before the drone again becomes dominant
2:30	Emergence of textural materials ending the first section
03:17-3:34	Descending harmonic textures presented in the foreground

Section 4.01 The Draw Tool/The Agent Tool V0.01

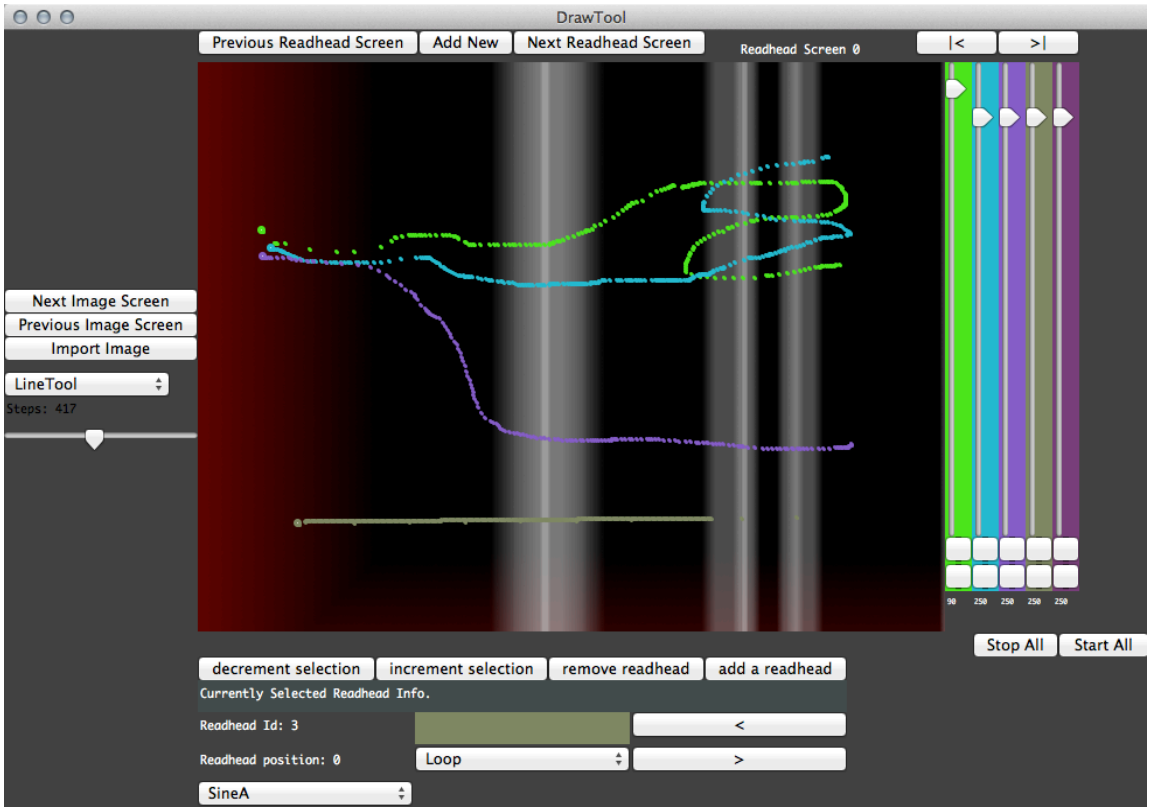

Figure 4.1 The Draw Tool/Agent Tool V0.01 Prototype

The creation of distinct sonic textures that develop with their own unique sense of internal motion was a key focal point in the composition of *Overture.* A minimum of three discrete drones would be performed live with each of these drones having independent speed and phrase lengths. While the level of activity and engagement with each of the textural drones varied, it was hoped that the cross rhythms and phase shifting of these materials would drive the piece forward. The software was designed with a voice-based architecture in which each constituent voice had its own sense of time to allow these compositional explorations to take place. This approach to composition and software design is mentioned by Vaggione (2001) and Dahan (2005) who both argued that time should not act as a shared constant but that tools should allow the traversal of various time scales, reflecting

the working methods of composers (Thiebaut et al., 2008). Unlike other graphic sound synthesis systems (see Section 3.01), image data was designed to be a source of stimulation and complement compositional materials, enabling users to draw paths imposed upon the image within the framework to create change. Image data is therefore used as a control stream within a parametric space rather than dictating the sonic output or *being* the sonic output. In a fashion similar to other voice-oriented graphical sound synthesis systems (notably UPIC and IanniX) the initial version of the Agent Tool made it possible for users to draw path-based image data that altered compositional parameters, the result not necessarily being perceived as a direct mapping. While it may be possible for these parameters to control global features of the work, or elements within (such as playback speed), they are conceptually designed to work within a user-defined structure, i.e. a score which the user has dictated. The system consequently performs a form of abstracted image sonification.

The Draw Tool was designed to be a small prototype that allowed the realisation of *Overture* and as such, the system was purposefully limited in scope. The prototype contained no internal synthesis engine with each voice emitting its control data via Open Sound Control (OSC) to a custom-built Pure Data patch which mapped the continuous data to appropriate synthesis parameters. The Pure Data (see Figure 4.2) synthesis framework made use of the University of Sheffield Sound Studio Tools (USSS Tools)[4] for rapid prototyping and exploration. Each voice in the Draw Tool was consequently designed to automate parameters of the granular synthesiser abstraction found within the tool kit. A voice within the system comprised this two-dimensional path across an image. When told to play, each voice emitted its position along with the Red, Green and Blue (RGB) colour components over OSC to the Pure Data synthesiser algorithm. As this data was emitted for external realisation, it was possible to explore a variety of different mapping

[4] The USSS Tools are a collection of Open Source compositional tools for a number of audio programming environments (Moore, A & Moore, D. 2010).

strategies. Aside from forcing the user to work with a two-dimensional voice based architecture, the prototype imposed no constraints as to how the data could be realised. Despite exploring a variety of different synthesis algorithms and mapping strategies as part of the compositional process of *Overture*, macroscopic control of a number of granular synthesisers proved most fruitful. This was in part because different loaded samples or live audio input could be used to create a huge variety of sonic materials without needing to alter the data mapping in any way.

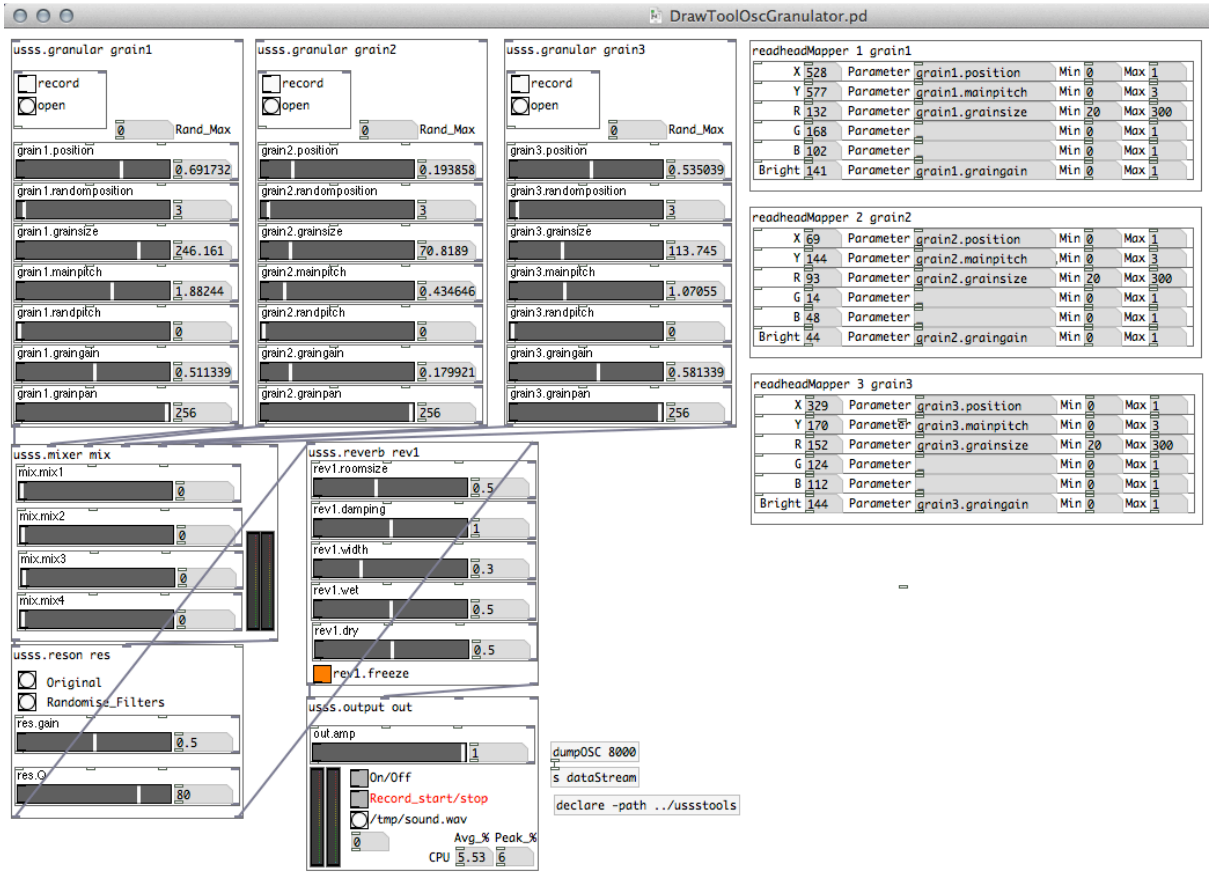

Figure 4.2 Pure Data granular synthesis framework designed for Overture using based around the USSS Toolkit

Throughout the composition of *Overture,* a number of favourable mapping strategies emerged through on-going exploration. These can be seen in Table 4.2.

Table 4.2 Colour mapping strategies explored in Overture

Data Source/Ma-pping ID	**X**	**Y**	**Red**	**Green**	**Blue**	**Brightness**
(a)	Position in sound file	Grain Pitch	N/A	N/A	N/A	Grain Stereo Spread
(b)	Position in sound file	Grain Pitch	Grain Size	N/A	N/A	Grain Amplitude
(c)	Position in sound file	Grain Pitch	Grain Size	Random Pitch Fluctuation	N/A	Grain Amplitude
(d)	Reverb Amount	Grain Pitch	Reverb Size	Reverb Damping	Reverb Width	Grain Amplitude

Across these explorations, grain pitch was consistently mapped to the vertical axis. This ensured predictable harmonic relationships. Strategy (c) allowed the green channel to influence random pitch fluctuations for variations. This strategy was only used briefly in the work to generate the contrasting agitated texture (03:41-04:02) building up to the gesture and change in space at 04:02. Strategies (a) and (b) were utilised prominently throughout the work. Approach (b) was used to generate the materials in the opening of the work (00:00-01:26) in conjunction with simple images that were created to vary the size of the grains being produced (see Figure 4.3).

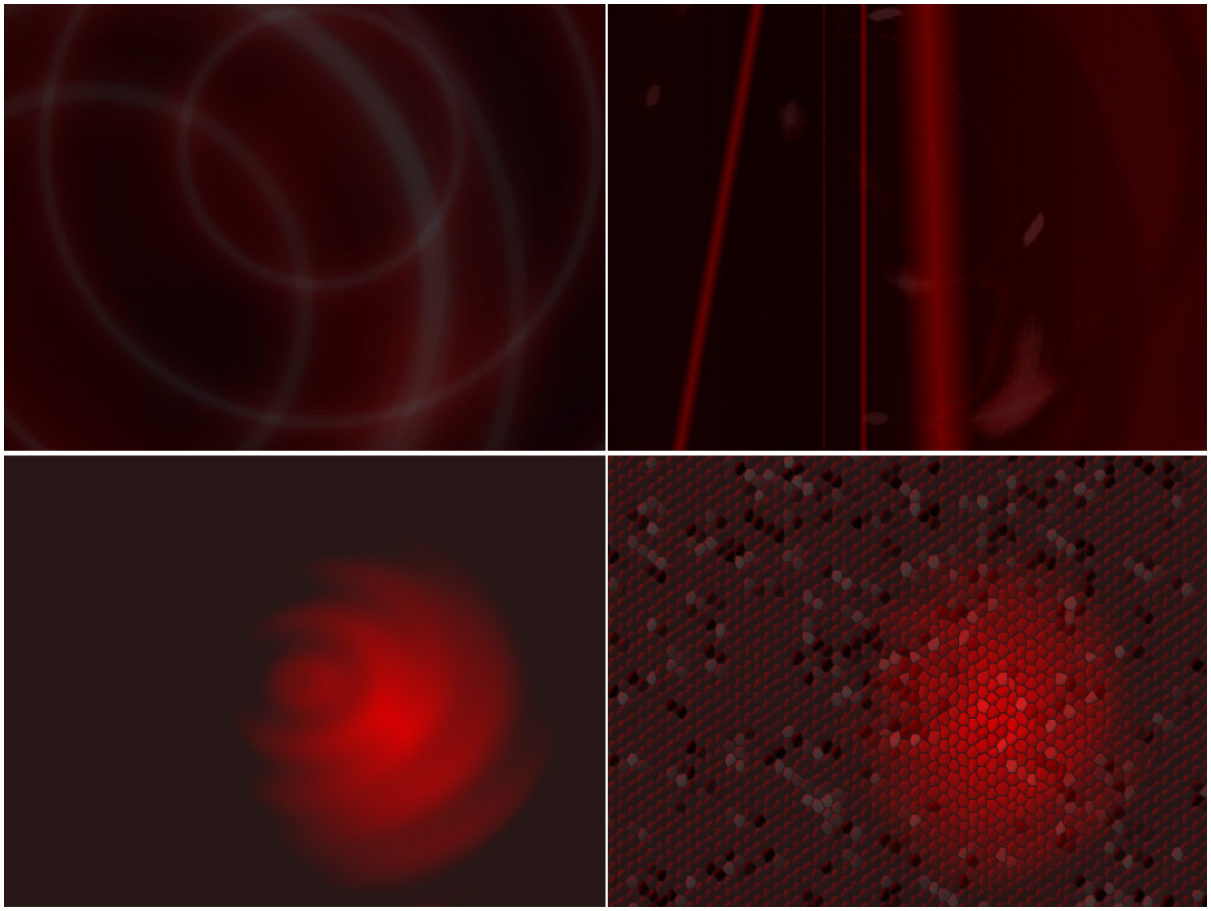

Figure 4.3 Images used to create variations in grain durations in Overture

While (b) was used to generate the dominant static drones in the work, (a) was utilised to generate contrasting materials that quickly move across the stereo field. This technique can be heard prominently in the in short noisy bursts of grains presented between 01:47-02:07 and 02:55-03:06. Using only pixel brightness and the red component enabled the impact of image data to be largely predicable. This approach behaved as a user bottleneck that subsequently resulted in more abstract sonifications. When more colour channels were used, the impact of the imagery on the synthesis algorithm made it harder to hypothesise the sonic output. The process of using all of the channels produced a highly diverse output of sonic materials that could have been used to generate complex musical gestures. As the work is primarily texture-carried, it was felt that the stochastic gestures that were generated using this technique should not be included in the work as

they may break the illusion of the work existing within a holistic and organic space. Strategy (d) was utilised to generate the short, heavily reverberated passage between 04:02-04:33.

Section 4.02 Evaluating the Draw Tool/Agent Tool V0.01

As a prototype system for traversing a parametric space in real-time, the Draw Tool was an effective system for the realisation of *Overture*. The macro level control over granular synthesis algorithms that utilised image data was effective as a tool for sonic exploration but was limited in scope. As the prototype was based entirely around controlling entire voices in Pure Data, a large computational overhead is placed on the system. The result of this overhead is that a limited number of voices can be performed in real-time which is evidenced by the usage of only three voices in *Overture.* At this point, the prototype relied too heavily upon a secondary sound synthesis system. It became important to consider controlling individual grains whilst not imposing specific mapping strategies. It was also important to retain an intuitive approach whilst offering greater complexity. Systems such as UPIC and IanniX have been used to explore Roads' concept of multi-scale composition (Roads, 2001) in which materials across a variety of timescales are explored within the same interface. Roads mentions that whilst holistic and multi-scale approaches are conceptually appealing, they can give unsatisfactory results on at least one level or time-scale (Roads, 1996). For example, controls for generating and controlling microscopic grains of sound will not necessarily be ideal for controlling meso or macro-time events due to the sheer number of entities that need to be controlled. If a system were designed to be open and ecosystemic in nature, could it be used to create an effective multiscale parameter space that feels intuitive whether working on microscopic or macroscopic timescales? Can imagery in such a framework be used as a means of mediating forms of stigmergy wherein entities influence one another? If so, can stigmergetic interactions facilitate parametric/timbral evolution across differing timescales? These questions arose from creative and computational research, drove future developments of the Agent Tool and produced numerous compositions, some working better than others.

(a) Timbre Space

Using imagery as a means of generating granular materials across a variety of timescales adheres closely to Xenakis' concept of a 'Timbre Space'. Timbre Space takes the form of a 'Vector Space' where, at any cross section of time within a work, all sonic entities can be represented as grains or 'cells'. Primarily this 'Vector Space' has three planes: frequency, amplitude[5] and time.

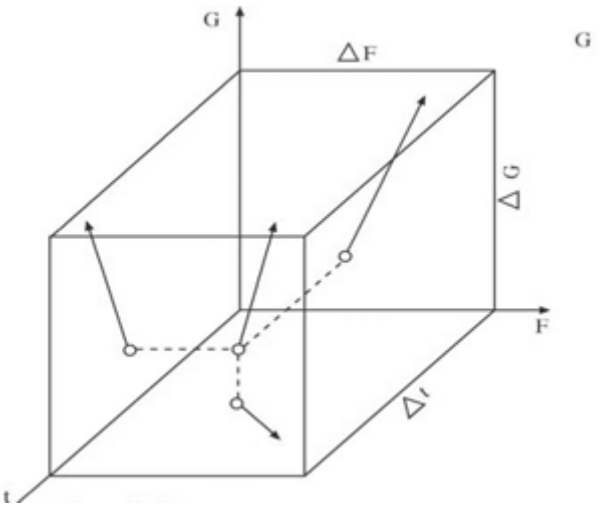

Figure 4.4 Xenakis' Vector Space Design (Xenakis, 1992)

This approach can be effectively achieved in a voice-orientated system with each discrete entity within the space corresponding to a given voice. Whilst materials can exist as separate entities within this space, their concepts of time are shared. This is to say that whilst each voice can alter its pitch and amplitude over time, voices traverse their paths through a shared concept of time. Attempts to overcome this issue were explored in the real-time version of the UPIC system (Marino et al., 1993). This has direct parallels with Roads' definition of multiscale composition in that it affords the ability to work across multiple time-scales, primarily the micro and macro (Roads, 2001). In this instance, it is possible for each part to have its own concept of time and for it to

[5] In Xenakis' case, amplitude is represented by performer intensity.

traverse it in a time-varying fashion[6]. This mapping of dimensions falls in line with Wessel's description of Timbre Space (Wessel, 1979), however, there is a clear difference in approach to the realisation of this sound. Whilst Xenakis' approach is firmly rooted in the exploration of microsound, Wessel's approach is deeply rooted within the usage of additive synthesis in collaboration with spectral analysis of existing instruments (Wessel, 1979).

Systems such as MetaSynth afford complex additive synthesis using image data to store the current Timbre Space. This approach allows quick access to and manipulation of vast quantities of spectral audio data which can be sculpted using simple image manipulation tools. Numerous sound manipulation techniques can be effectively explored through intuitive image manipulation.

The approach of Xenakis and Wessel can be summarized thus.

- Xenakis: Realizing objects and the relationships of objects within a 'conceptual' space. Using data in an abstract form (sonification)
- Wessel: Realizing the entire space. Using data in a concrete form (audification).

Furthermore, image data in was not fully explored in these systems[7]. Red Green, Blue and Alpha channels were often neglected or averaged, turning the image into a grey-scale representation with the amplitude of a given frequency proportional to pixel brightness. The Agent Tool as used in *Overture* was designed to allow grain-based realisations of this Timbre Space akin to

[6] A direct consequence of this is that it allowed independent parts to speed up and slow down independently as one would expect in an instrumental setting.

[7] MetaSynth allows the exploration of Red and Green colour components in mapping these colours to the spatial positioning of a given frequency.

Xenakis' definition. It also highlighted the compositional appeal of grain-based exploration of image data.

In relation to the concept of a Vector Space, Xenakis proposed that at any given time interval ΔT, grains can be represented on a two-dimensional screen with the axis ΔF and ΔG (representing frequency and density respectively). Consequently one can traverse through these screens either to generate a sonic output or to analyse a sonic input. These screens can subsequently have a form of elementary operations performed on them. The intersection of two screens can produce a shared cloud of grains. The union of screens can produce a cloud based on materials in all child screens. A complement screen produces a screen made of all grains that are not shared across child screens. Finally, the difference can be represented by subtracting grains from another screen.

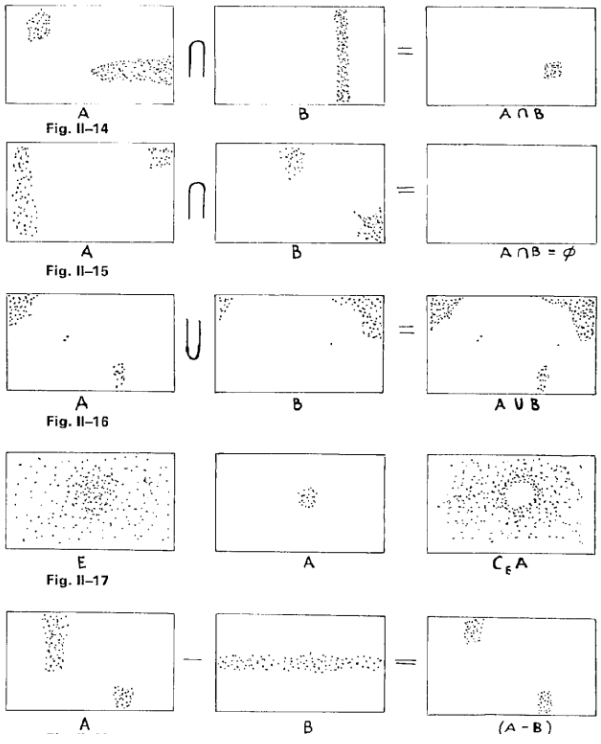

Figure 4.5 Xenakis' screen manipulations (Xenakis, 1992)

With this approach to screens in mind, agents exist as entities within a construct whereby each frame of animation represents a new screen and a screen represents a snapshot of a timbre

space. Static imagery can be viewed as a screen representing a static view port into a virtual image or space. Video represents a collection of sequential screens or timbre spaces. In the Agent Tool there is a clear separation between imagery and agent data. Agents, exist in an entirely separate space from the subsequent image stimuli and can even read data from imagery that may not be on screen. Ecosystemic grain exploration of these screens subsequently became the focus of the work *Modular Void* and led to the development of the first significant iteration of The Agent Tool.

Chapter 5. The Agent Tool V0.02 and *Modular Void* (2013) – 32:00

The first significant version of the Agent Tool was completed during the early stages of composing the work *Modular Void*. The Agent Tool from this version onward was designed to be a complex and reusable composition environment that afforded the scripting of abstract agents of varying complexity to control elements of synthesis and sound manipulation. The system is underpinned by a micro-threaded architecture which allows a user to define, spawn, manipulate and terminate agents from small scripts which can be invoked in real-time. The micro-threaded architecture is designed to allow the creation of complex synthesis and control mechanisms from simple scripts that are written from the perspective of a performer/agent within the system. Consequently, complex control mechanisms and ecosystems can be crafted in an efficient manner without requiring a significant amount of programming knowledge. The Agent Tool's AI system utilises a form of co-operative multitasking rather than the commonly used Finite-State Machine (FSM). Co-routines are used to allow each AI to operate in its own pseudo-thread without the resource cost of system threads (Dawson and Entertainment, 2001).

The system was designed to be an open-ended framework whereby all agent data could be emitted via Open Sound Control (OSC) for realisation externally thus imposing as few constraints on the user as possible. Whilst the system can be used to control any audio software that accepts OSC, the micro-threaded design of the system works in a fashion similar to the Node-based architecture of SuperCollider. The system is optimised for this package and features a collection of tools which can dynamically spawn and manipulate nodes on a remote SuperCollider server. It also contains an OSC server which allows external software packages and controllers to manipulate the properties of *any* instantiated agent. This facilitates a unique form of object-oriented composition and interaction (see Section 5.01(b)). The framework allows the usage of static imagery, recorded video and real-time camera input as a means of stimulating agents within the system. Usage of imagery is not

imposed upon the user, but can be used as a means of controlling and invoking agents (see Section 7.01(a) and Section 8.01). Imagery is also designed as a means or invoking stigmergy whereby agents can influence the performance of others via altering or drawing on the underlying image canvas (see Section 6.02).

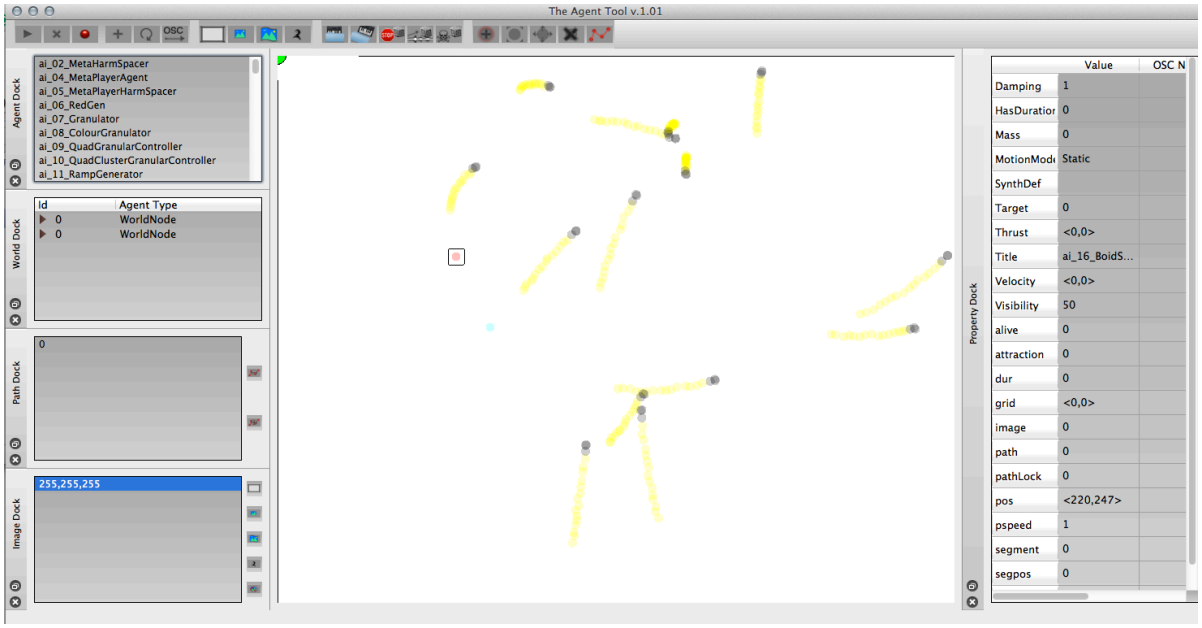

Figure 5.1 The Agent Tool Graphical User Interface

Section 5.01 Implementation and Technical Notes

The Agent Tool was developed using C++ with the Qt Windowing System. All agents within the system can have their behaviour scripted by a user using an embedded form of the scripting language Lua (Ierusalimschy et al., 1996). A comprehensive collection of Lua functions and data types are provided alongside appropriate documentation with the software installer. The accompanying data DVD contains installers for both Mac OS X (10.7+) and Ubuntu (12.04+). The package contains a suite of agent scripts that the user can begin using upon booting the system. In this instance, a suite of specific examples is provided. Both Agent scripts and subsequent SuperCollider synthesiser definitions can be found in a folder called *TheAgentTool* which should be placed in the user's home directory upon running the package for the first time.

(a) Lua Integration

The Agent Tool's AI system utilises a form of co-operative multitasking rather than the commonly used Finite-State Machine (FSM). Each agent within the system runs in its own co-routine/pseudo-thread. Within the system, a single Lua state exists. When an agent is created, it is spawned within its own thread that belongs to the parent Lua state. Upon commencing playback, the Lua state goes through each agent 'brain' that is active and executes its code accordingly (see Figure 5.3). However, as Lua state itself is not multi-threaded, it executes each sub-thread sequentially. This means that unless a brain-script indicates that it is finished, for the time being, the system will become caught within that thread. To overcome this, the user must allow other co-routines a chance to process, either by calling a Lua function 'coroutine.yield()' or an Agent Tool function 'pause( )' (Figure 5.3). A collection of core functions within the system (such as spawning a SuperCollider node) will automatically yield. To create a brain-script, the user simply needs to create a Lua text file and create a function that represents what the agent should do when asked to run. This function should be prefixed with 'ai_' and call a function '\initai()'[8] from within it (see Figure 5.2 for the equivalent of 'hello world'). Upon loading a Lua script, the system parses all of the text within it and identifies all functions with the prefixes 'ai_'. The interface is then updated to reflect this, showing the names of all available functions and consequently allows the user to select which agents it wishes to spawn.

[8] 'initai()' acts as means of ensuring that the thread does not immediately start running upon its spawning

```
function ai_HelloWorld()                    1
  initai()                                  2
  addP("freq",220)                          3
  addP("amp",0.8)                           4
  setP("SynthDef","default")                5
  pause(1)                                  6
  die()                                     7
end                                         8
```

Figure 5.2 A basic agent who plays a single note upon spawning

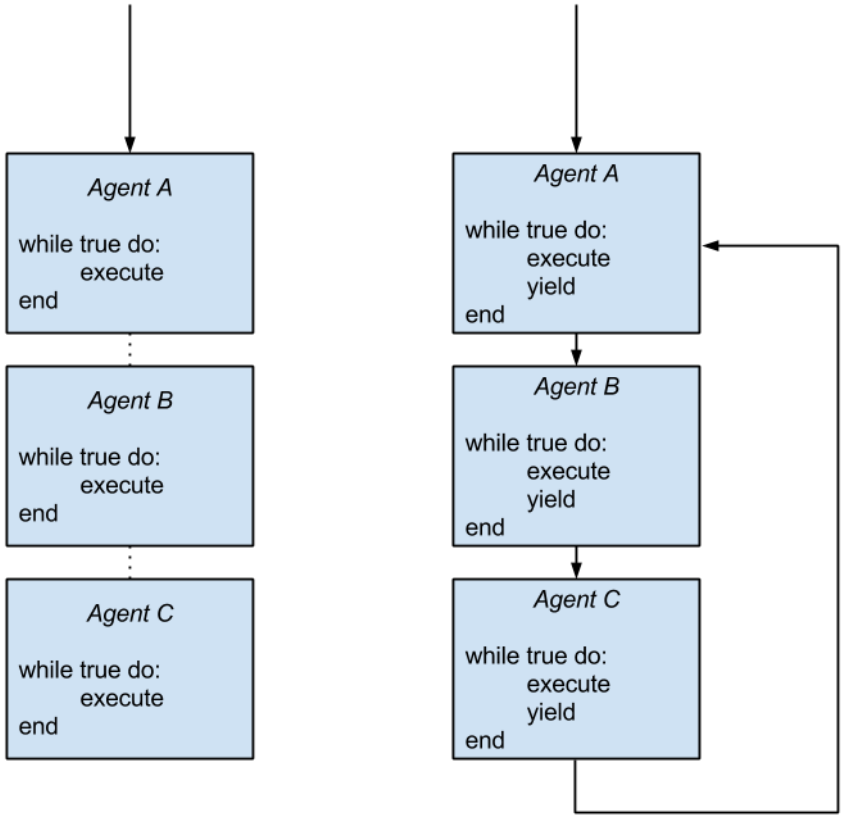

Figure 5.3 Example blocking and non-blocking agent script

The Agent Tool provides a rich set of features on top of the Lua Application Programming Interface (API). In addition to functions controlling and manipulating agents and imagery, it also

contains a fully featured two-dimensional vector class and RGBA colour class that can be utilised for the creation of streamlined two-dimensional maths and colour manipulation.

An agent can also request image data from within its component script. All forms of imagery are stored within a 'screen database' which the user can traverse. Within this context, each 'world' can have a complete database of useable imagery. The system is designed to allow multiple 'worlds' to exist at any given time. Currently, only one instance is permissible. 'Worlds', similar to agents, can be spawned, controlled, manipulated and destroyed from within the scripting engine. Consequently 'worlds' can be viewed as an auxiliary means of grouping agents together within the system. Within these constructs, Agents are constrained to the bounds of its parent 'world'. Upon activating the 'physics' motion model (see Section 5.01(f)), agents will react to the gravity and bounds of its parent world. Furthermore, in this context, agents will utilise thrust, mass, damping and energy loss components so that they can subsequently 'rebound'.

Image data may be obtained in three distinct forms from within the system:

1. From the pixel at an agent's current position.
2. Via an intermediate agent. If the identifier (id) of an agent is known, it is possible to obtain associated colour data at its location.
3. Via direct access to pixel data at specified co-ordinates.

Across all three instances, it is possible to obtain red, green and blue components along with the total and the average colour weighting. Given these three interfaces and the relationship model within the system, highly complex *abstract* realisations of graphic sound synthesis can occur.

(b) TreeNodes, WorldAgents and Agents

Within the system three core object types exist: TreeNode, WorldAgent and Agent. The TreeNode acts as the base class for the system for both WorldAgents and Agents alike. These subclasses can, therefore, be viewed 'as' TreeNodes with specialised feature sets (Meyers, 2005). The GUI for the

system in turn, allows a user to show and interact with the features of any TreeNode within the system. Consequently, both Agent and Worlds can be viewed within the same interface. Their parent-child relationships are visualised within the framework.

The purpose of the TreeNode class is six-fold:

1. Provide a standardised interface for graphical user interaction
2. Manage object life-cycles
3. Store and manage properties
4. Manage SuperCollider connections
5. Manage the life-cycles of child nodes
6. Manage an object's motion model

It is possible for any TreeNode within the system to have an active lifecycle or duration. The TreeNode base class is responsible for managing the life and death of a node/agent. The duration/lifetime of a TreeNode can be set by itself, by another TreeNode or by graphical user interaction. At any given time a TreeNode can obtain the time it has been alive and how long until it is scheduled to be terminated.

(c) Properties

A property within the system represents a unified form of storage for data of different types. Properties can be used to represent booleans (true or false states), integers, floating point numbers, text or two-dimensional vectors. A TreeNode (including WorldNode and Agent) features a container of properties which can be accessed both within the GUI and the scripting engine alike. Within both contexts, the user has the ability to get and set the property in question. Properties do not just represent an element of data however. Every instance of a property contains a set of internal flags that indicate whether the user can read, write and/or send this data directly to SuperCollider. By

default, every node within the system is constructed with a suite of properties (and controls) of differing types. These include controls for motion simulations and lifetime management. The scripting API allows an Agent to add, get, and set its own properties (addP, getP and setP). In the case of adding and setting, entire Lua tables can be provided which contain key-value pairs of data (i.e. freq=440). Across all instances, alternative forms exist in which the function can be performed on any Agent if its ID is known (addPID, getPID and setPID). Example usage of the property system from the scripting environment can be seen below in Figure 5.4.

```
function ai_PropertyExample()                                          1
  initai()                                                             2
  addP("Hello","world")                                                3
  local extraProperties = {A = 1, B = Vec2.new(10,10), C= 3.14159 }    4
  addP(extraProperties)                                                5
  setP("A",13)                                                         6
  local id = getID()                                                   7
  setPID("Hello","information",id)                                     8
  pause(5)                                                             9
  die()                                                                10
end                                                                    11
```

Figure 5.4 A basic agent script illustrating a variety property interactions

(d) SuperCollider Integration

Each instantiated node within the system stores an accessible unique identifier. This ID is used to control a node running on a SuperCollider server if requested. In this case, the life and death of a node on a SuperCollider server is dictated by the system. Furthermore, if a property's flag is set to 'send' when its value is changed, the subsequent SuperCollider node is sent the new value. Each TreeNode in the system monitors when a user 'sets' a property. To initialize this connection to

SuperCollider, a property entitled 'SynthDef' is monitored[9]. When the 'SynthDef' is either set or added, a SuperCollider node will be spawned with the TreeNode's ID. All properties that exist within that TreeNode that can be sent, are then bundled into the construction arguments for a SuperCollider Node. By default, when one creates a property, data is sent to the appropriate address with the same name. For example, if a property called 'BandWidth' is created, whenever it is updated, the data that this property is storing will be bundled into an appropriate OSC message with the name 'BandWidth'. The working sequence of events can be seen in Figure 5.5.

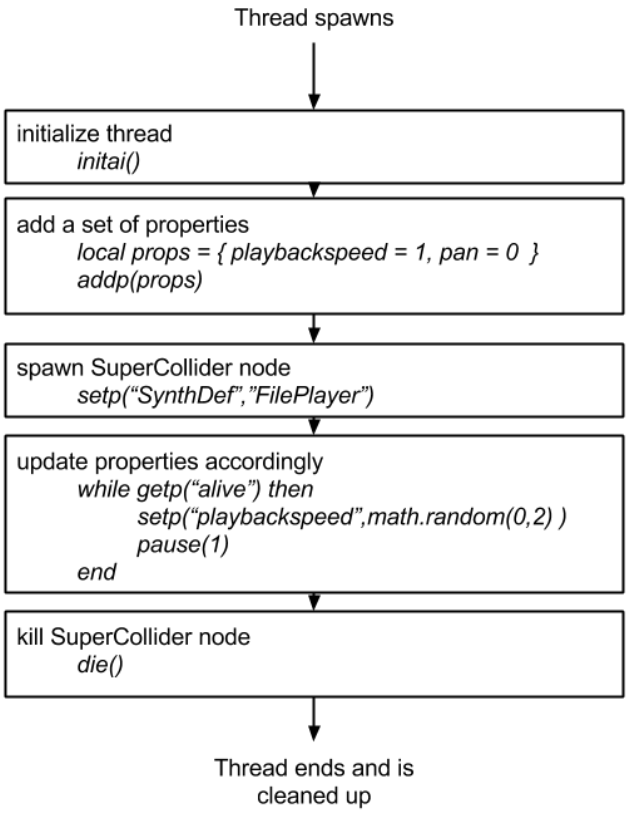

Figure 5.5 A typical agent life cycle illustrating general SuperCollider usage in pseudo code

[9] For convenience the property is named 'SynthDef' to mirror its name within the SuperCollider environment

Each property has an optional storage for a personalised OSC path. If this is set, when the value is updated, the data will be sent via that path instead. The user can specify the IP Address and port number for this connection so that it can connect to remote instances of SuperCollider or alternative audio packages. From within the scripting engine, there are commands that are provided specifically for usage with SuperCollider, 'scNew','scSet' and 'scFree' *(see* Figure 5.7*)*. These allow the manual spawning of SuperCollider nodes, setting an argument/control within a node and the 'free-ing' (deletion) of a Node. Similarly, a set of basic functions to send raw OSC data exist. These functions allow the sending of integers, floats and strings of text. The 'send' function can be used to send any data to a specified IP address, port and path.

```
function ai_SendOscExample()
  initai()
  send("127.0.0.1",9000,"/hello", "pure data", 3.14159)
  setP("Velocity",Vec2.new(10,10))
  for i = 1, 5 do
    local pos = getP("pos")
    send("127.0.0.1",9000,"/position", "x", pos:x(), "y",pos:y())
    pause(1)
  end
  die()
end
```

Figure 5.6 An agent script illustrating how OSC data can be sent to external packages

The Agent Tool allows a user to store a path to a folder containing a collection of compiled Supercollider synthesisers. If the user indicates to the framework where this folder is, the system will connect to SuperCollider and invoke it to load these instruments automatically. A set of basic pre-built synthesisers that come with the system can be used, resulting in the user not needing to program in SuperCollider if they so wish. Alongside a collection of example scripts, the framework

similarly provides a small collection of pre-compiled SuperCollider instruments which can be found in the TheAgentTool folder in the user's home directory. The system also contains GUI functions to boot, clear and shutdown a remote SuperCollider server. It also allows users to graphically load sound files into SuperCollider buffers at indexes that they can specify. Consequently, it is possible for a user to utilise SuperCollider without necessarily needing to program or invoke code within it.

(e) Relationships

Active agents are afforded the ability to spawn and manipulate child agents that they may or may not assume responsibility for. The framework imposes no direct limit on the number of potential children that can be spawned. However, spawning thousands of agents that each invoke sound from SuperCollider will potentially lead to system slowdown. It is up to the user and system's discretion to impose a limit on the number of agents to spawn. At any time, an Agent has the ability to spawn other agents of any given type. To add a child, either the functions ‘spawn’, ‘spawnRelative', ‘spawnP' or ‘spawnRelativeP’ can be called from within the scripting engine (see Figure 5.7). In the case of the latter two functions, a table of properties needs to be provided to establish or set properties of a child agent from within the parent. All of these functions return the Id of the child that has been spawned for usage within the scripting engine at a later time. These spawned agents can exist as children of the parent agent, children of an alternative agent or they can be assigned to exist simply within the current ‘world’. All agents consequently have the ability to manipulate the properties of their children. Parent agents may wish to monitor the properties of child agents accordingly. This approach enables musical features such as amplitude, spatial positioning and harmonic content of child agents to be studied by a parent. Upon monitoring such features or properties, related agents can influence those whom they have access to. For example, a parent agent could be utilised to ensure that child agents are adhering to a set of pitch strata.

```
function ai_SpawnExample()
  initai()
  local childProps = {A = 1,B = "Hello", C = Vec2.new(100,50)}
  local id = getID()
  spawnRelative(id,1,"ai_Emptygrain",0,0)
  spawnRelativeP(childProps,id,1,"ai_Emptygrain",10,10)
  pause(5)
  die()
end
```

Figure 5.7 An agent script illustrating how to spawn relative Agents

As agents do not implicitly create audio, it is possible to utilise hierarchical relationships within the framework to create 'control' agents who create, manipulate and destroy scores of agents through a variety of means. These control agents can therefore be used as a means of creating effective instruments and interactions. This approach is highly effective for creating granular synthesis frameworks. In this context, a single control agent can act as the primary interface spawning hundreds of child agents at discrete time intervals which represent individual grains. It is entirely up to the user to decide what these child agents should be, whether they should generate sonic material, or whether they themselves merely control other agents or systems. This affords the realisation of complex, yet efficient granular synthesis systems (see Figure 5.8 and Figure 5.9) and recursive compositional ideas (see Figure 5.10 and Figure 5.11).

```
function ai_BasicGrain()
  initai()
  local pos = getP("pos")
  local worldsize = getWorldSize()
  local props = {freq =pos:y(),pan = pos:x() / worldsize:x(),amp = 0.2,
      dur = 0.1}
  addP(props)
  setP("SynthDef","default")
  while true do
    pos = getP("pos")
    setP("freq",pos:y())
    setP("pan", ((pos:x() / worldsize:x())*2)-1 )
  end
  die()
end

function ai_BasicGranular()
    initai()
    addP("rate",0.01)
    setP("Velocity",Vec2.new(50,0))
    local id = getID()

    while true do
      spawnRelative(id,1,"ai_BasicGrain",0,0)
      pause(getP("rate"))
    end
    die()

end
```

Figure 5.8 An agent script representing a grain and a control Agent who continuously spawns these grains

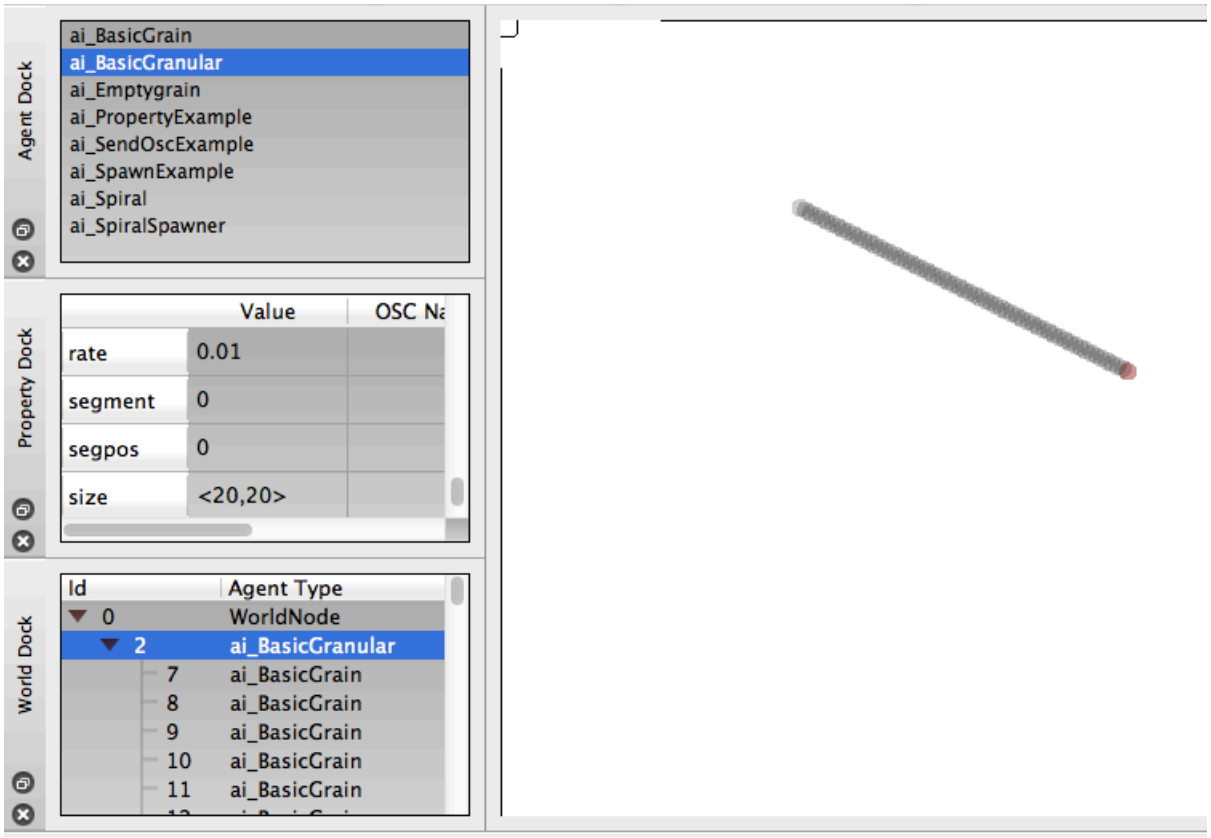

Figure 5.9 The resultant realisation of ai_BasicGranular

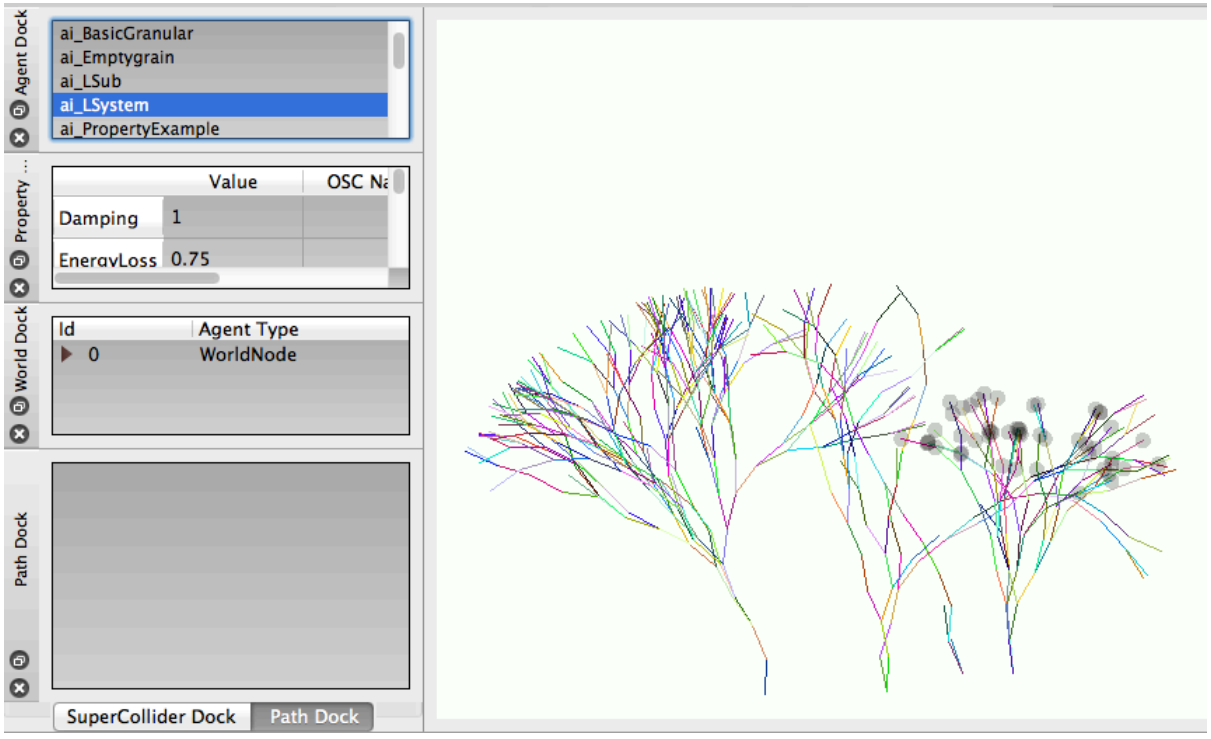

Figure 5.10 A resultant realisation of ai_LSystem.

```
function ai_LSystem()
  initai()
  local childProps = {depth = 12, theta=3.141592}
  local pos = getP("pos")
  spawnRelativeP(childProps,0,1,"ai_LSub",pos:x(),pos:y())
  die()
end

function ai_LSub()
  initai()
  pause(0.01)
  local pos = getP("pos")
  print(pos)
  local args = {freq = x,SynthDef = "default"}
  addP("freq", (pos:x()+pos:y())/2.0 )
  addP("amp",0.01)
  setP("SynthDef","default")
  if getP("depth") > 0 then
    pause(0.1 + math.random()*0.2)
    if math.random() > 0.3 then
      local t = ((math.random()*2 )- 1) * 0.5 + getP("theta")
      local cPos = pos + Vec2.new(math.sin(t)*25,math.cos(t)*25)
      local childProps = {depth = getP("depth")-1,theta = t}
      drawLine(currentScreen(),pos,cPos,math.random(10,255),math.random
          (10,255),math.random(10,255))
      spawnRelativeP(childProps,0,1,"ai_LSub",cPos:x(),cPos:y())
    end
    if math.random() > 0.3 then
      local t =((math.random()*2 )- 1) * 0.5 + getP("theta")
      local cPos = pos + Vec2.new(math.sin(t)*25,math.cos(t)*25)
      local childProps = {depth = getP("depth")-1,theta = t}
      drawLine(currentScreen(),pos,cPos,math.random(0,255),math.random
          (0,255),math.random(0,255))
      spawnRelativeP(childProps,0,1,"ai_LSub",cPos:x(),cPos:y())
    end
  end
  die()
end
```

Figure 5.11 An example recursive L-System using a pair of agent scripts

This approach to relationships and the unified parametric control of properties enables an agent to represent an entire generative musical structure. Controls can be efficiently created and manipulated through the property interface. Alternatively, these properties can control elements of musical form. As entire structures of sonic agents can be invoked from single instances, these instances themselves can be utilised within a sequence of scripted events. As such, the composition process moves away from sound design to more formal score creation.

Agents within this context can become components of a generative musical system and form finite sonic objects. This difference between the *generative* and *finite* is reflected in different approaches to the ownership of an agent's lifetime and existence. When any agent is spawned from within the scripting interface, the unique identifier of a parent agent must be provided. When an agent dies, it kills all of the children for which it is responsible. Consequently to efficiently terminate an entire generative system, the user only needs to destroy the controlling parent agent. By default, this ensures that all sonic structures exist in a finite form. Similar frameworks such as Khorwa (Malt, 2004) adapt a more generative approach to an agent's existence. In the context of Khorwa, a parent agent generates offspring which then exist detached from the parent. A result of this is that when the parent dies, the offspring remain and continue to perform tasks. Such an approach results in a complex generative system that is comprised of many discrete agents with hundreds of controls. Due to the polymorphic relationship between WorldNodes, Agents and their base class, TreeNode, it is entirely possible to create purely generative systems within The Agent Tool. When a parent ID is provided upon the spawning of an agent, this ID must belong to a TreeNode, i.e., it can belong to either an agent, or a world. Therefore, for Agents to exist as truly independent entities, the ID of a world can be provided (the default world has an ID of 0).

The parent-child relationship in The Agent Tool enables a single agent to spawn a variety of sounds and control materials with timing and position data that is entirely relative to its existence.

This approach can be used to generate groupings of musical materials across different time scales and represent clouds of grains, sonic objects, musical phrases, passages or even an entire work. From the user's perspective, the underlying scripting approach remains the same: spawn a child agent, wait for a period of time, spawn another agent etc., before dying and killing off all child agents (if they themselves haven't already finished).

(f) Motion Models

To enable agents to traverse the parametric space with predictable and repeatable motions, every agent within the system contains a construct known as a 'Motion Model'. It is the purpose of these 'Motion Models' to dictate the way an agent moves within its parent world. The first iteration of the Motion Model system contained three core 'models': static, linear and physics. After the completion of *Modular Void* and preliminary sketches for the composition *Furcifer,* the path model was added to explore convergence and divergence through a parametric space (see Section 6.02). When using the static motion model, real-time control over the Agent's velocity and calls to 'move()' in the scripting engines will be ineffective. The linear motion model gives access to the node's velocity property and also scripted motion in the Lua engine. The physics motion, on the other hand, combines a variety of world and agent properties to calculate movement. World properties such as gravity, uniform energy loss and world size are combined with an agent's thrust, mass, velocity and damping to calculate an artificial physics simulation in which agents can be effected by gravity and interact with the edges of the 'world'. Gravity within the framework is a two-dimensional construct which can influence agents in both axes.

Section 5.02 *Modular Void* (2013)

Modular Void is a suite of three works composed between May and November of 2013. The work is an exploration of 'dronescapes' generated from an assortment of granular synthesis systems created within the Agent Tool. The suite can be performed as a whole or as individual movements.

The first two movements contain materials that are all structurally linked, with content being introduced and developed before the movement concludes. The third movement contains a number of references to previous movements in its final section. *Movement One* and *Movement Two* received their premiere at the International Festival for Innovations in Music Production and Composition 2014 in Leeds.

The work explores the usage of grain-based image sonification and audification using the Agent Tool. Movements evolve from being texture-carried to gesture-carried, ultimately resulting in entropic gestures. The work moves from tightly knit clouds of grains generated from explicit real-time user control to granular synthesis systems that relinquish user control in favour of forms of abstract image sonification.

As a 'dronescape' piece, the work focuses upon the subtleties of spectral motion and internal dynamics of each component sound object. The primary textures that make up the work function to "demonstrate internal behaviour without indicating direction" (Stavropolous, 2006, p.253), whilst streams of micro-sound serve to add contrast and suggest further development. Harsh and dissonant harmonic textures emerge out of key structural gestures. Similar to *With What Remains* and *Malfunction 30931* by Gough (2008) and Thigpen (2011) respectively, the base source materials are drawn from computer errors and glitches that have been rendered to produce usable audio. Initially, these materials are heard indirectly with heavily processed forms serving as a pedal for the first two movements. As the listener travels through the work, more and more of these original recordings are gradually exposed as the work's gestural intensity grows.

Section 5.03 Relative Granular Synthesis and Meta-Controllers

Across the entirety of *Modular Void*, a variety of original granular synthesis systems were created within the Agent Tool, based around relative object/agent-orientated scripting. The composition explores three key features of Agent Tool functionality.

1. Real-time control and performance of agent-based granular synthesis algorithms. Simple control mechanisms based around an agent's properties relative to the world around it and other agents.
2. Image audification and sonification as a means of influencing and enhancing these algorithms.
3. Tracking agents within the system.

This final feature was explored late in the compositional process and became a key area of interest for future works (notably *Liten Röst* and *Mikro Studie A*). Throughout the compositional process of this work a body of 50+ agents were created who were in turn invoked and controlled in real-time as a means of exploring sonic material. A large number of these agents had their scripts altered in real-time whilst sonic output was recorded[10]. The examples that follow are a collection of agents that were all used and modified within *Modular Void*.

(a) Granular Synthesis Meta Controllers

As mentioned previously in the section *Relationships*, the parent/child hierarchy within the Agent Tool makes it highly efficient to programme *controller* agents that do not generate sound. As evidenced in Figure 5.8 and Figure 5.9 this approach works well with granular synthesis algorithms as the user can create an agent who goes on to spawn other agents (with varying durations). Timing and positioning within the parametric space are controlled through the creation of appropriate properties within the parent controller agent. In the case of both the parent agent and its child agents, any of their component properties can be controlled by the sonification or audification of image data.

Across the entirety of the compositional portfolio presented here a large suite of both audible and control agents were developed. All of the audible agents utilised their own independent mapping and playback mechanisms. A collection of agents that invoked the playback of a single sound file in SuperCollider were created. These agents would then map the amplitude, panning,

[10] Live coding within the framework will in turn be discussed further in 0.

duration, the sound file to play, speed of sound file playback and start position of playback to different properties such as world coordinates, pixel brightness or the value of underlying colour components.

The composition of *Modular Void* also saw the development of a number of different mapping strategies (of position and colour) for 'audible' grain agents as shown below.

Table 5.1 Mapping strategies employed in Modular Void

Data Source	**Parametric Mappings**
RGB colour at agent's position	Grain position in sound file, grain window shape, grain size, grain pitch
Vertical / horizontal position	Grain position in sound file, amplitude, grain pitch and panning

Mapping the position in this instance felt somewhat familiar compositionally, but using colour data required extensive experimentation in order to generate subtle textural materials. This was partly due to a necessity to become familiar with the relationship between the colour palette of an image and the spectromorphology of the sound file being granulated. One favourable mapping strategy that emerged in the early experiments for *Modular Void* was to map the start position of sound file/grain playback to the horizontal position within the world.

When controller agents spawned at very high rates moved across the horizontal axis at different speeds, differing time stretches of the source material would emerge. This led to the creation of a suite of grain agents that utilised this mapping in conjunction with strategies for the vertical position and colour components.

Modular Void makes full use of harmonic transformations of drones. The opening of the first movement is a key example of this. This movement is driven by a largely static drone on C# spread across several octaves. Variations of this drone, a tritone below and a perfect fourth above are utilised to contrast and drive the work forward (heard in the low bass at 01:30 and developed

again at 03:30). To achieve materials that contained relative harmonic content, a second common mapping strategy for grains emerged, this time mapping pitch to vertical position. A typical example of such a grain can be seen in Figure 5.12 with pitch being altered via changes in playback speed.

```
function ai_GrainA()
  initai()
  local pos = getP("pos")
  local worldsize = getWorldSize()
  local props = {speed =pos:y()/261.625 ,startpos = pos:x() / worldsize:
      x(),amp = 0.5,bufNum = 0 }
  addP(props)
  setP("SynthDef","grain")
  while true do
    pos = getP("pos")
    local col = getRGBA(currentScreen())
    setP("speed",pos:y()/261.625)
    setP("startpos", pos:x() / worldsize:x() )
    setP("amp",math.pow( getMeanCol(currentScreen()) /255,2 ))
    coroutine.yield()
  end
  die()
end
```

Figure 5.12 ai_GrainA used as a blueprint for grain development

'ai_GrainA' uses the greyscale average of the current pixel to control the amplitude of the current grain. This rather simple approach to sonifying colour data allows the influence of an image's content to be predictable. This predictability allows imagery to be used as a performance space which can be easily traversed in real-time. In composing *Modular Void* these agent grains were invoked and controlled manually on a number of occasions. Timing information for these usages can be found below in Table 5.2. In the opening movement, this agent type is used to

generate short (30ms in duration) grains over 10Khz which stand in contrast to a pre-existing long harmonic drone. Swells in amplitude were generated through the creation of imagery comprising bands of white on a black background (see Figure 5.13).

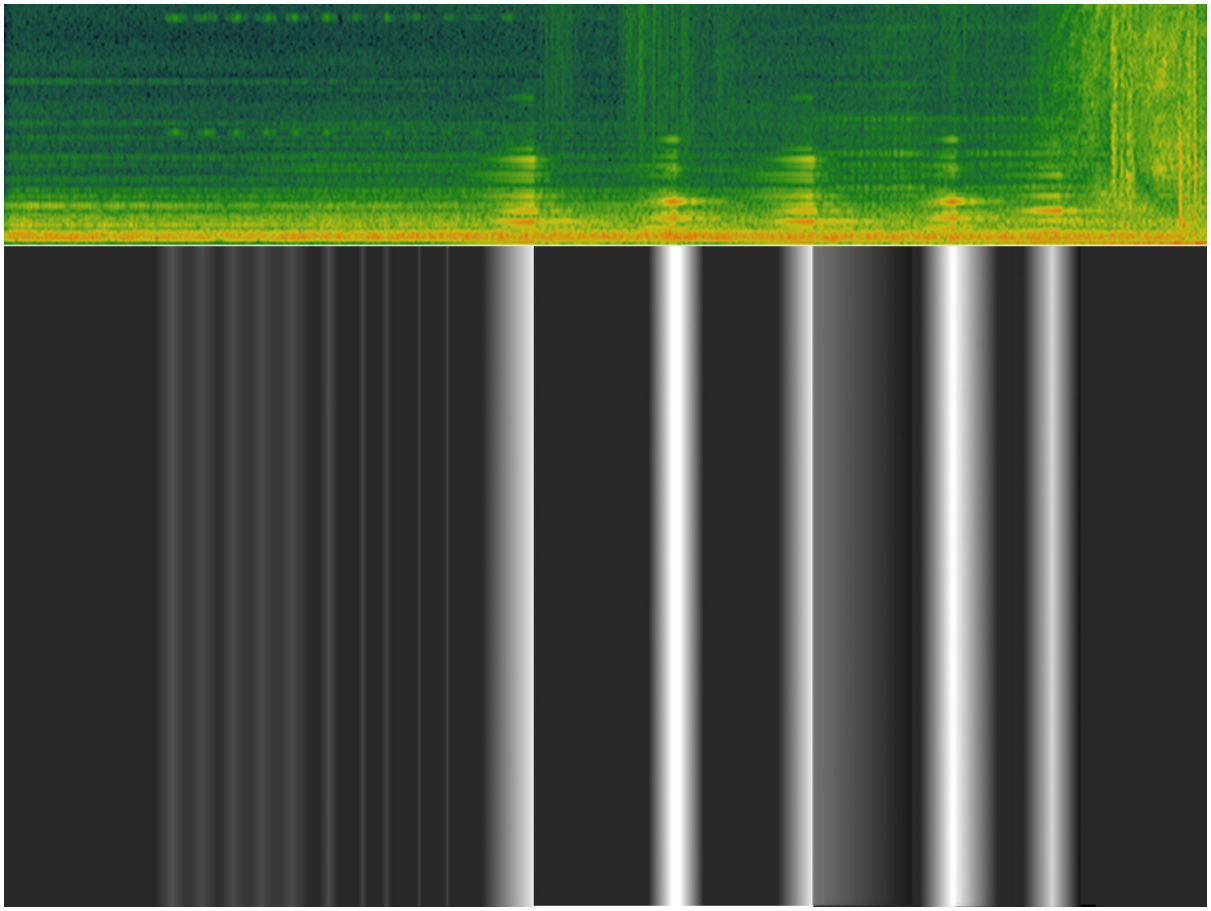

Figure 5.13 Black and White barcode used as part of the opening of Modular Void Movement Two (00:00-01:30) alongside the spectrogram of the passage

In this instance the movement of the spawning agent was controlled manually. The sonic output of these grains is designed to introduce the listener to the noisier and more gestural materials that follow in subsequent movements. This approach to using bands of colour as a means of generating amplitude control data takes on more of a significant compositional role in the following movements. In *Movement Two* this technique was used to generate a series of harmonic swells which are developed and reprised throughout.

Table 5.2 ai_GrainA usage in Modular Void

	Movement	Timing	Description
Ai_GrainA	1	02:55 – 03:19	Short high frequency (10Khx+) bursts approx. 30ms in duration
	2	00:30 – 01:10	Harmonic swell phrase
	2	02:17 – 03:30	Harmonic phrase variation A
	2	04:00 – 04:55	Harmonic phrase variation B
	2	08:24 – 09:42	Harmonic swell phrase reprise
	2	11:30 – 12:04	Variation A reprise

In *Movement Three* an alternative approach to amplitude control was taken as a means of generating more erratic and aggressive variations in amplitude. In this instance, individual RGB components were used to control the respective volume of three contrasting sources of noise. This gave rise to some highly dynamic sonic material that acted as a precursor to an aggressive final passage (10:30-11:20). One of the agents used can be seen in Figure 5.21. Similar to the techniques used on *Divide by Zero* (Thigpen, 2007), raw binary files are used as sonic input, generating a harsh and abrasive sound world. These sources of noise were then presented at 01:43-02:12, 02:20-03:02 in *Movement Three* and 00:24-01:25 in *Movement Two*. In the closing passage of the work, imagery constructed from strict bands of colour were used (see Figure 5.15) which generated rapid and repeatable mix automation.

Throughout the composition of *Modular Void* two different approaches to grain based image sonification were explored. In the first of these forms, the agent continuously updates the audio engine of choice throughout its life cycle as seen previously in 'ai_GrainA' and 'ai_PlayerMorpher'. In these instances, agents act as augmented controllers for sound file playback no matter what their duration. Within this context, grains perform continuous audification or sonification of the underlying data. The alternative approach sees agents invoke an appropriate sound file when appropriate colour data is detected. From this perspective agents can perform event-based image sonification. A comprehensive RGBA interface, at this point not developed, began to be explored to aid the development of the composition *Mikro Studie A* (see Chapter 8).

```
function ai_PlayerMorpher()                                                    1
  initai()                                                                     2
  local props = {bufAAmp = 0.3, bufBAmp = 0.3,bufCAmp = 0.3,dur = 10,mix       3
      = 0.5,room=0.5}
  addP(props)                                                                  4
  setP("Velocity",Vec2.new(100,0))                                             5
  setP("SynthDef","playerMorpher")                                             6
  while true do                                                                7
    local worldSize = getWorldSize()                                           8
    local position = getPosition()                                             9
    local colour = getRGBA(0)                                                  10
                                                                               11
    setP("bufAAmp",colour:r() / 255)                                           12
    setP("bufBAmp",colour:g() / 255)                                           13
    setP("bufCAmp",colour:b() / 255)                                           14
    setP("mix",position:x()/worldSize:x())                                     15
    setP("room",position:y()/worldSize:y())                                    16
                                                                               17
    if position:x() >= worldSize:x() then                                      18
      setP("pos",Vec2.new(0,position:y()))                                     19
    end                                                                        20
    coroutine.yield()                                                          21
  end                                                                          22
                                                                               23
end                                                                            24
```

Figure 5.14 ai_PlayerMorpher used in Modular Void and extensively in Furcifer in which RGB components control the mix of three sound files

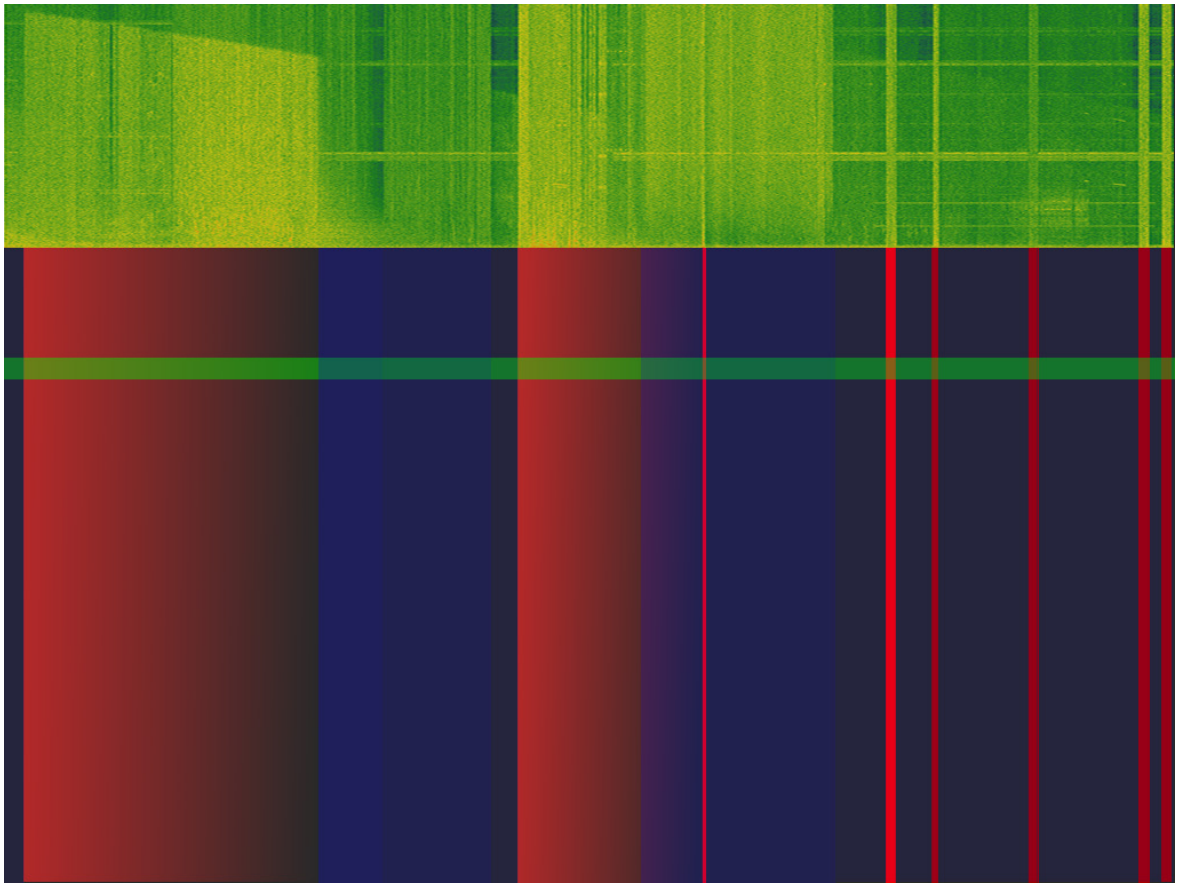

Figure 5.15 Barcode image used to automate mixing of three audio streams in Modular Void Movement Three (10:22-11:22) alongside the spectrogram of the passage

(b) Relative controls and logic

Alongside a suite of grain agents, a collection of spawning controller agents were created and, similar to Figure 5.8, agents were spawned at a given rate. Grain position randomisation (as seen in Figure 5.16 in and Figure 5.17) was a key feature of such agents. The majority of child agents had their position in the sound file mapped to the horizontal axis and their pitch to the vertical axis. This allowed both subtle and extreme transformations of materials. Large passages in *Movement Two* were created by applying minuscule pitch fluctuations to harmonically pure materials, resulting in amplitude modulation through beating. Many of the drones presented in the opening of this movement (00:00-01:20) were derived using this technique. Real-time control over 'xRandom' and 'yRandom' properties of the basic 'ai_Granulator' (see Figure 5.16) generated high-frequency noise, the first occurrence of which is at 00:11. It was then used extensively as a sound generation device.

This agent was also used to spawn instances of 'ai_GrainA' in *Movement One* as discussed previously in Granular Synthesis Meta Controllers.

```
function ai_Granulator()                                                    1
  initai()                                                                  2
  local id = getID()                                                        3
  local props = {rate = 0.01, childAgent = "ai_GrainA",xRandom = 3,         4
      yRandom = 3,childDur = 0.5}
  addP(props)                                                               5
  setP("Velocity",Vec2.new(20,0))                                           6
  changeMotionModel("Physics")                                              7
  while true do                                                             8
    local xRandom = getP("xRandom")                                         9
    local yRandom = getP("yRandom")                                         10
    local x = math.random(xRandom* -1,xRandom) * 0.5                        11
    local y = math.random(yRandom* -1,yRandom) * 0.5                        12
    spawnRelativeP({dur = getP("childDur")},id,1,getP("childAgent"), x,y    13
        )
    pause( getP("rate"))                                                    14
  end                                                                       15
end                                                                         16
```

Figure 5.16 ai_Granulator agent that acted as a template for other granular synthesis systems

In this instance, random pitch variations were altered through text input in the user interface. (The speed of alteration was dictated by how fast a user can input variations in text!) As a result of this limitation a variation of this basic granulator was created (see Figure 5.17). 'ai_ColourGranulator' utilises the red and green colour components of a pixel to alter the random variations of child positions in both dimensions. A series of images were consequently created to automate the random child positions. This controller was subsequently used to generate the

harmonic swells throughout *Movement Two* (00:00-00:33) via the creation of images with thin vertical bands of green (see Figure 5.19) which 'ai_ColourGranulator' read from left to right.

```
function ai_ColourGranulator()
  initai()
  local id = getID()
  local props = {rate = 0.01, childAgent = "ai_GrainA"}
  addP(props)
  setP("Velocity",Vec2.new(20,0))
  changeMotionModel("Physics")
  while true do
    local mean = getMeanCol(currentScreen())
    local col = getRGBA(currentScreen())
    local x = math.random(col:r()* -1,col:r()) * 0.5
    local y = math.random(col:g()* -1,col:g()) * 0.5
    spawnRelativeP({dur = (mean /255)*2},id,1,getP("childAgent"), x,y)
    pause( getP("rate"))
  end
  die()
end
```

Figure 5.17 ai_ColourGranulator used to spawn and alter grain displacements in space based upon red and green colour components.

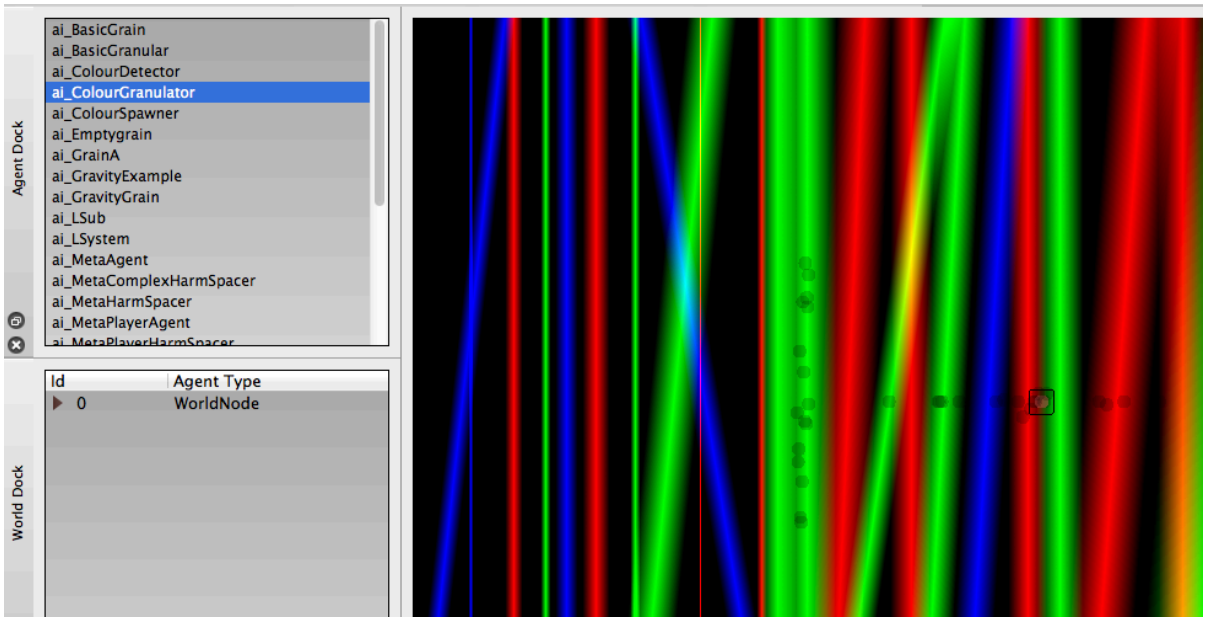

Figure 5.18 Two instances of ai_ColourGranulator. Note the vertical and horizontal displacements of child agents.

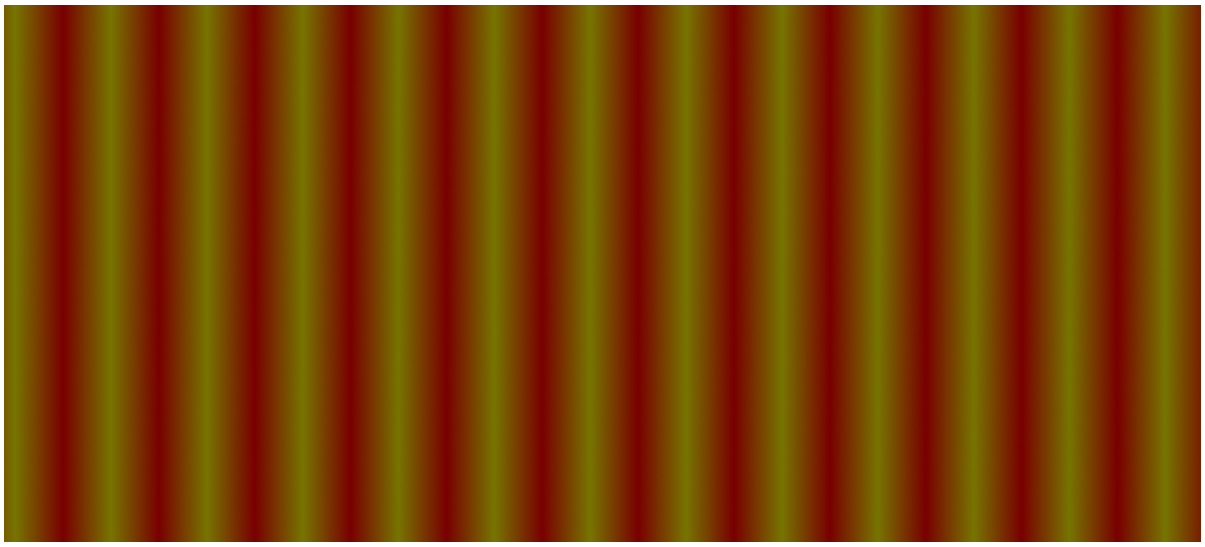

Figure 5.19 Green band imagery used to generate the opening pulsing *drone of Modular Void Movement Two*

Towards the end of the compositional process, focus shifted from using agency to create grains and controllers to the creation of meta-controllers. These were used frequently to impose harmonic structures onto drones. The opening drones of *Movement One* and *Movement Three* were constructed using 'ai_QuadGranularController' (see Figure 5.20 and Figure 5.21). These drones are dominant in the opening of *Movement One* but are presented in distal space in the opening of *Movement Three*. This agent encapsulates four 'ai_ColourGranulators' and affords real-time control of pitch and position information. Drones were generated by setting the granular agents at finite vertical spacings. The quad controller was then used to slowly traverse the source file at a slow rate, giving rise to time-stretches with imposed pitch content that was influenced by imagery. This approach was also used in *Liten Röst* (the opening of *Movement One* and 02:20-02:40 of *Movement Two*).

Relative controls in the system were used to explore implosion and explosion around set points in the parametric space. This can be heard throughout the composition of *Movement Three* (see Figure 5.24). This approach to movement and clustering of agents in a parameter space

Figure 5.20 A graphical representation of ai_QuadGranularController

generates pronounced forms of parameter convergence or divergence across a number of controls. Whilst this was theoretically appealing, these experiments were not used in the final composition as the mappings were difficult to implement in such a way as to be musically useful. Convergence/divergence led to binary mappings and the point of interest in these experiments remained static. The point of interest could dictate spatial positioning or the position within a sound-file: altering these once instantiated, is somewhat cumbersome when using the text entry system in the UI. This shortcoming informed the development of the path system utilised when composing *Furcifer* (see Chapter 6). One of the more favourable mapping strategies that emerged was the mapping of grain position in the parent sound-file to the x-axis and the pitch of these grains to the y-axis respectively. In this parametric space, 'ai_StackSpawn' (an agent which explored relative movement and spawning in time - see Figure 5.22) generated ascending/descending sequences of pitch transformations similar to pitch accumulation transformations. These time-pitch variations consequently contrast greatly with the static drones that are the focus of the work.

```
function ai_QuadGranularController()
    initai()
    local space = 30
    local props = {run = 1, rate = 0.01,ChildAPos = Vec2.new(0,space
        *-2) , ChildBPos = Vec2.new(0,space *-1),ChildCPos = Vec2.new(0,
        space ) , ChildDPos = Vec2.new(0,space *2)}
    local id = getID()
    addP(props)
    local childA = spawnRelative(id,1,"ai_ColourGranulator",getP("
        ChildAPos"))
    local childB = spawnRelative(id,1,"ai_ColourGranulator",getP("
        ChildBPos"))
    local childC = spawnRelative(id,1,"ai_ColourGranulator",getP("
        ChildCPos"))
    local childD = spawnRelative(id,1,"ai_ColourGranulator",getP("
        ChildDPos"))

    setP("Velocity",Vec2.new(3,0))

    while getP("run") > 0 do
      local pos = getP("pos")
      setPID("pos",getP("ChildAPos") + pos,childA)
      setPID("pos",getP("ChildBPos") + pos,childB)
      setPID("pos",getP("ChildCPos") + pos,childC)
      setPID("pos",getP("ChildDPos") + pos,childD)

      pause(getP("rate"))
    end
    die()

end
```

Figure 5.21 The meta-granular synthesis controller used in Modular Void and Liten Röst

```
function ai_StackSpawn()                                                    1
  initai()                                                                  2
  local props = {dur =5, childTotal = 10,stackDistribution=4,stackHeight    3
      = 400,childType = "ai_GrainA",run = 1}
  addP(props)                                                               4
  local id = getID()                                                        5
  while getP("run")>0 do                                                    6
    local args = {dur = getPID("dur",id)}                                   7
    local total =getPID("childTotal",id)                                    8
    local height = getPID("stackHeight",id)                                 9
    for i = 0, total ,1 do                                                  10
      local ratio = getPID("stackDistribution",id)                          11
      local child  = spawnRelativeP(args,id,1,getPID("childType",id), 0,    12
          math.pow(i/total,ratio) *height )
    end                                                                     13
    pause(getPID("dur",id))                                                 14
  end                                                                       15
  die()                                                                     16
end                                                                         17
```

Figure 5.22 ai_StackSpawn designed to generate stacks/accumulations of other Agents

Figure 5.23 Several instances of ai_StackSpawn employed in Modular Void

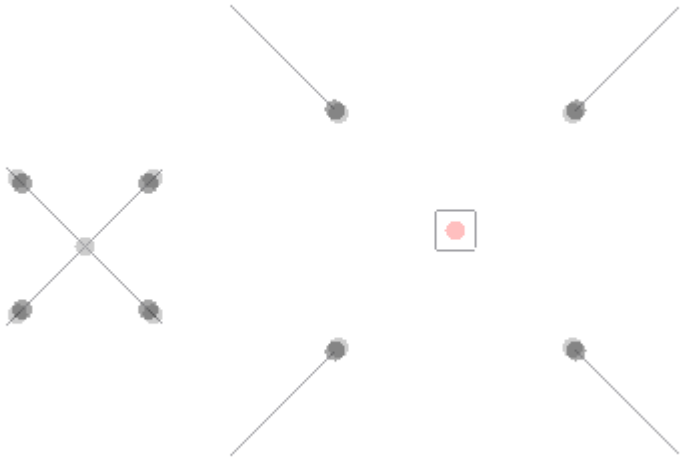

Figure 5.24 Graphical representation of colour granulators imploding and exploding explored in the creation of Modular Void

(c) Primitive Agent Tracking

As part of the compositional process for *Modular Void* a number of initial experiments were undertaken in which agents track/monitor the properties of other agents. As the parametric space is open, the tracking of elements can have a variety of compositional metaphors[11]. The impact of such tracking has the potential to be directly audible or a component within a control structure. As the logic for the tracking of properties exists within the scripting engine, the routing and usage of these streams can vary over time. Within the system an agent can have access to any property currently extant if the ID of the parent agent and property name is known. Throughout writing *Movement Two* variations on the implosion system were being developed as a means of generating complex glissandi. Initially this was done through the creation of a property representing gravity (two-dimensionally) in the WorldNode which agents then monitored. This property would then be manipulated in real-time to impose motions throughout the current parameter space. This

[11] The metaphors for tracking agents, whether explicit, implied, audible to the listener or simply visible to the composer was often musically beneficial.

mechanism was consequently utilised by a suite of agents scripted manually. To optimise this process the *physics* motion model was devised to make this accessible to all agents within the system (see Section 5.01(f)) with their own unique masses. As motion models can be set through the property interface, they too can be tracked and altered.

In the final version of the composition glissandi are used as a means of generating tension prior to gestural events in *Movement Two*. 'ai_GravityExample' was used to achieve this on a number of occasions (see Figure 5.26 and Figure 5.25), most notably between 05:07-05:28 and 8:03-08:17 in extensive upward glissandi.

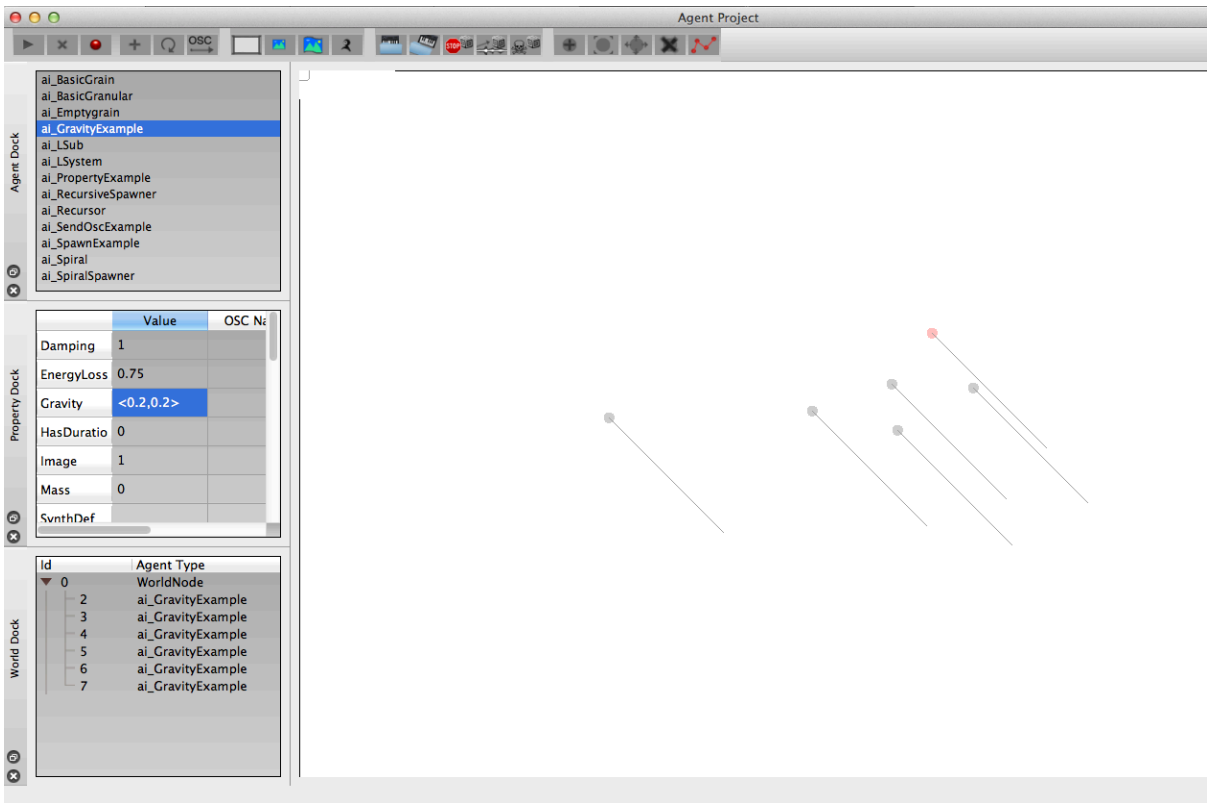

Figure 5.25 Several instances of ai_GravityExample under the influence of gravity. Their velocities are represented by grey lines

```
function ai_GravityExample()
  initai()
  changeMotionModel("Physics")
  setP("Mass",0.001)
  local pos = getP("pos")
  local worldsize = getWorldSize()
  local props = {bufNum = 0 , pan =(  (pos:x() / worldsize:x()) * 2.0)
      -1, rate = pos:y()/261.6255,dur = dur,amp = 0.1 }
  addP(props)
  setP("SynthDef","player")

  while true do
    pos = getP("pos")
    worldsize = getWorldSize()
    local amp =  math.pow( getMeanCol(currentScreen()) /255,2 )
    setP("amp",amp)
    setP("pan",(  (pos:x() / worldsize:x()) * 2.0) -1)
    setP("rate",pos:y()/261.6255 )
    pause(0.01)
  end

  die()
end
```

Figure 5.26 The sound file player Agent whose rate of playback and position is influenced by gravity used in Modular Void

As the position of an agent can be obtained through the property interface, agents can directly track the position of one another within the parameter space. Across the research presented here this basic mechanism is used in a variety of ways, culminating in *flocking* and *predator prey* simulations used throughout *Liten Röst* and *Mikro Studie A* (see Section 7.01(c) and Section 8.01). In *Modular Void* position tracking was purposefully one dimensional.

'ai_HorizontalFollower' (see Figure 5.28 and Figure 5.27) was used to generate a number of high frequency canopies across the work. These materials are presented at the close of the *Movement One* and feature prominently in *Movement Two* and *Three*. Musical interest and development of these canopies is created through the cyclic loops and artificial beating generated from pairs of these agents continuously seeking to converge on one another.

Table 5.3 Usage of ai_HorizontalFollower in Modular Void

	Movement	Timing
Ai_HorizontalFollower	1	04:41-05:30
	2	00:11-00:25
	2	01:20-02:05
	2	03:04-05:09
	2	05:52-07:57
	3	02:43-05:00

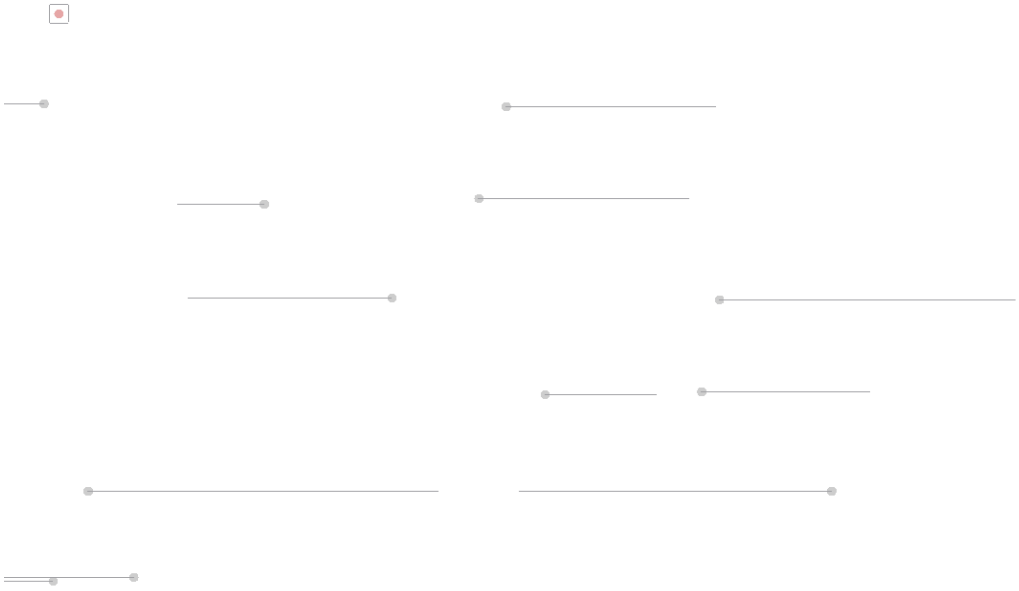

Figure 5.27 A Graphical representation of ai_HorizontalFollowers being used on Modular Void

```
function ai_HorizontalFollower()
  initai()
  changeMotionModel("Physics")
  local pos = getP("pos")
  local worldsize = getWorldSize()
  local props = {Mass = math.random(0.01,0.6),amp = 0.4,speed =pos:y()
      /261.625 ,startpos = pos:x() / worldsize:x() ,attraction = 1.2,dur
       = 30}
  addP(props)
  setP("SynthDef","grain")

  while t < dur do
    pos = getP("pos")
    setP("speed",pos:y()/261.625)
    setP("startpos", pos:x() / worldsize:x() )
    setP("amp",math.pow( getMeanCol(currentScreen()) /255,2 ))

    drawCircle(currentScreen(),pos,3,RGBA.new(255,0,0,150),-1)

    local targetId = getNearest(pos)
    local targetPos = getPID("pos",targetId)
    local dir = (targetPos - pos):norm()

    setP("Thrust", Vec2.new(dir:x() * getP("attraction"),getP("Thrust"):
        y()))

    coroutine.yield()
  end
  die()
end
```

Figure 5.28 The horizontal follower agent used to create clustered canopies in Modular Void

Section 5.04 Evaluation of Agent Tool V0.02 - *Modular Void*

Throughout the creation of *Modular Void* it became clear that the biggest strength of the framework was the efficient creation of complex granular synthesis algorithms and controllers. The combination of the open *parent-child* relationship model of the system and the micro-threaded nature of *Lua* became one of the system's key strengths. As there is a separation between the spawning agent and the child agent, the script for audible grains can be generated and modified whilst the parent agent is spawning them. This allows grains to be altered rapidly for sonic exploration and the embellishment of existing materials. Grains can also be enhanced to suit a specific source, sound or image. Altering agents in real-time worked well with extended textural materials.

The usage of imagery as a stimulus for spawning agents and grain agents is highly perceivable, and can be used to generate both audifications and sonifications within the same framework. Imagery as a form of parametric automation on both levels can yield highly predictable results if the image data being used is constructed with a somewhat simple colour content.

Similarly, when subtle gradients or imagery with repeating patterns were used with this agent, the influence of imagery was noticeable. Using imagery with dense changes in pixel content yields largely unpredictable sonic outputs. Imagery within this context consequently gives rise to stochastic behaviours. When this approach to explicit pixel mapping is combined with granular synthesis algorithms, huge variations in musical output can be generated.

Throughout the composition of *Modular Void* attempts were made to create agents that traversed the parameter space in a set way. Whilst it is possible to script specific motions, crafting an agent that moves to a series of points in the space is achievable via the creation of a finite state machine within the agent's script. This approach allows a user to be highly explicit in the motion of the agent, but it is rather time-consuming to implement within a single agent and is dependent on

significant programming experience. This issue is made more cumbersome when image audification or sonification is involved as the points, and the motions have to be re-scripted for each image source that is imported into the system. To mitigate this, attempts were made to create repeatable patterns of entropic materials via scripting 'ai_PlayerMorphers' to move in and around materials with dense pixel colour changes. This form of automation through the parameter space became a key inspiration for the work *Furcifer*. Subsequently a system for creating, editing and deleting paths efficiently within scripts and on the graphical interface was devised.

Chapter 6. The Path System and *Furcifer* (2013) – 09:32

Furcifer was completed in early 2013 and received its premiere at the 'From Tape to Typedef' conference at the University of Sheffield. The term *Furcifer* is a reference to a genus of chameleon based on the Latin term *Furci*, meaning forked (a reference to their feet). As a chameleon changes its colour according to mood, so *Furcifer* uses spectral colour change and spatial references to invoke different emotions in the listener. The work was composed around a number of key sonic gestures and textures that were generated via the creation of paths through a parameter space within the Agent Tool. As such the composition demonstrates the use of path generation and control to sculpt sonic gestures and textures.

The work is in an *ABA* form with the *B* section exposing the listener to a primarily surreal space and the surrounding sections exploring real and unreal source-space relationships. Both *A* sections feature a dialogue between what Wishart called *unreal objects/real space* and *real objects/unreal space* (Wishart, 1996). Similar to Bayle's *Erosphere* (Bayle, 1992) and Gobeil's *Point De Passage* (Gobeil, 2001), the metaphor of a door is used at key points in the work to indicate entering a new space.

Furcifer relies heavily on the listener's perception of recordings of common sonic landscapes and phenomena as a means of invoking tension. Similar to works such as Stollery's *Still Voices* (Stollery, 2007) and Harrison's *Undertow* (Harrison, 2007), the sonic landscape regularly morphs between what Harrison describes as *real*, *unreal* and *surreal* spaces (Harrison, 1999) and this becomes a structural device. Amongst unreal sonic objects, the sound of (real) doors is significant as it is one of a few clearly source-bonded sonic objects (Smalley, 1997). As Windsor and Kendall discussed (Windsor, 2000; Kendall, 1991), the usage of everyday sound events afford the perception of not only the event that triggered the sound but also recollections of what traditionally follows this event.

There are several types of source-bonded materials that are used throughout the work. These include recordings of orchestral strings, orchestral percussion and recordings of the sung and spoken voice. Several of these have defined functions throughout the work, either as clear structural reference points or as a means of embellishing unreal spaces with real objects. The majority of these sources are transformed and mixed through agents traversing a path which performs parametric automation.

The key thematic sonic object that opens the work appears in several variations at the end of *A* (03:01), at the start of the reprise (06:45), re-harmonized (07:15) and marks the start of the closing passage (07:55). These variations were purposefully devised with the path system. The path system was also used to create a series of ongoing glissandi. This key thematic object is used as a metaphorical door, signifying transitions into new spaces throughout the work. Similar to the works *Overture* (see Chapter 4) and *Liten Röst* (see Chapter 7) the human voice was used as a source for the creation of textural materials and more harmonic content. Structurally, the voice was used as a means of supporting important events and phrases. For example, elements of human breathing can be heard in the main sonic object of the work and acts as an unreal backdrop when the first real space (a forest) is heard around 01:40.

Upon commencing work on *Furcifer,* The Agent Tool was adapted to enable the establishment of paths through current parametric space. Unlike systems such as Swarm Lake (Kaliakatsos–Papakostas et al., 2014) where an agent's relationship with a path is binary, agents in the Agent Tool are attracted to or repelled by paths. This approach was adopted so that attraction can be altered in real-time by a user or via scripting the 'thrust' property. Detailed mechanisms involved and examples of use will be discussed in Section 6.02. As with other elements within the Agent Tool, path modification can be created and manipulated in both the GUI and the scripting

engine. Allowing the scripting interface to create, move and delete both paths and points means that paths can be algorithmically manipulated as part of a work.

Section 6.01 Path Implementation

The path system can be accessed from within an agent via the 'Path' motion model. The usage of the system can therefore be scripted. A path within the framework consists of an array of points which agents can interpolate. The space between two points is referred to as a segment. A path comprised of three points [a,b,c] subsequently contains two segments [a-b] and [b-c]. When utilising the path system, an agent will traverse along a segment until it reaches the target point at the end of the segment. Upon doing so, the agent will then shift to the next segment (see Figure 6.1).

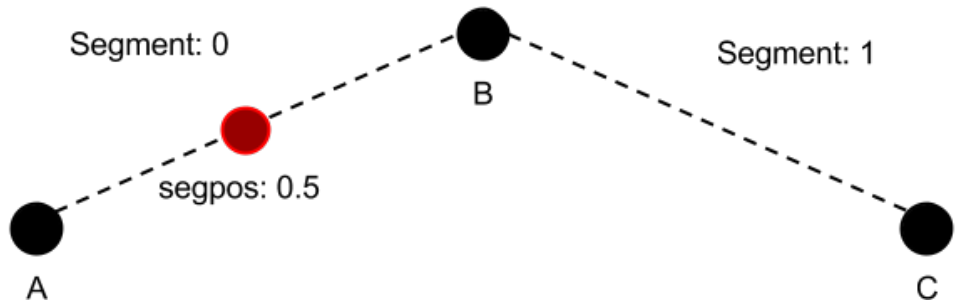

Figure 6.1 The breakdown of an agent on a path containing three points

To do this, the Path motion model utilises a set of properties that are stored within its parent agent. A direct result of this is that both the user and the scripting engine have the ability to alter directly the way in which the agent traverses the path. For example, upon prompting, the scripting engine can enforce the looping of segments or the traversal across segments (in a prescribed or random manner). The system relies on six properties being present for full functionality. A summary of these are as follows.

Table 6.1 Summary of path related agent properties

Property Identifier	**Function**
path	The ID of the path it is currently interacting with
segment	The segment of the path the agent is on
segpos	The agent's position on the current segment
pspeed	The speed of an agent's traversal along a path
pathlock	Snap an agent's motion to the current path
thrust	Two dimensional attraction/repulsion to current path

The system in place allows agents to interact with the paths to varying amounts. An agent may either be locked to a path, or influenced by a path's presence (allowing both attraction and repulsion). The 'pathLock' property allows this interaction to be fixed or not. When 'pathLock' is off, the system utilises the 'thrust' property to represent the strength of attraction (in two dimensions) to its relative position on the path. As a direct consequence of this, it is possible for agents to drift between path points, shifting in and out of coherence with the path that it is following.

Depending on the mapping strategy employed by an agent, paths can become highly perceivable with interactions being clearly audible. Mapping pitch to one axis of the world is the most obvious approach. Within this context, paths can be used to set out chords or clusters of notes. Increasing the attraction factor of the agents will consequently force them to adhere more and more to this pitch strata. In the context of spatialization, paths can be used to represent spatial trajectories of materials whose influence can vary over time.

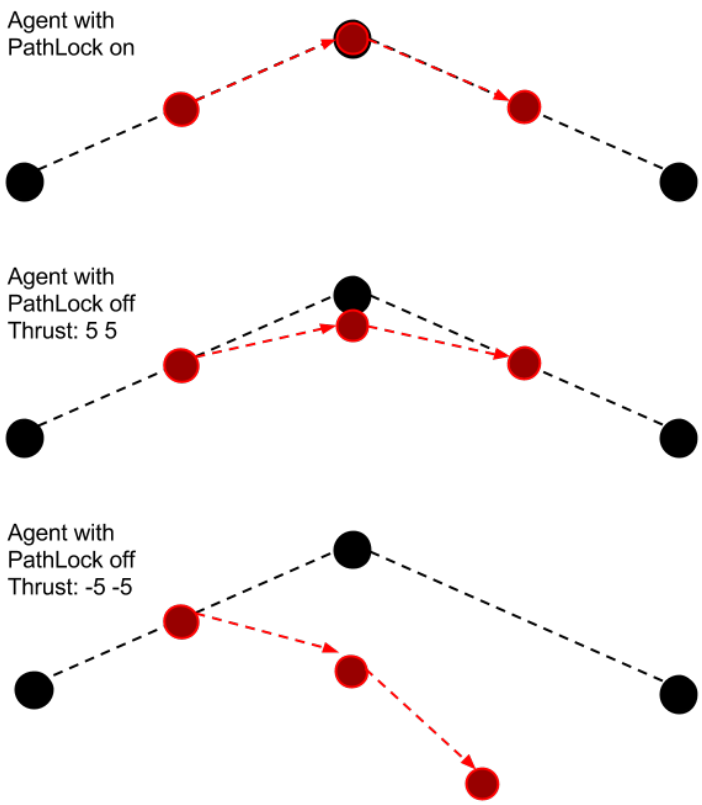

Figure 6.2 Comparison of path lock alongside positive and negative thrust

Section 6.02 Agent Explorations in *Furcifer*

The agents used in the composition of *Furcifer* are all based on iterative and evolutionary developments of an agent explored previously in the work *Modular Void.* Alongside a basic granular synthesis controller (see Section 5.03), the work uses four further agents. Two of these agents investigate position-based agent tracking in a parametric space. The first of these, 'ai_trackWorld' is designed to track the position of the current WorldNode and advance towards it continuously (see Figure 6.3 and Figure 6.4). When using these agents as part of the compositional process, the present WorldNode's motion model was set to *path*. An assortment of paths were then drawn and modified whilst these agents were tracking the WorldNode. This approach allowed the indirect creation and influence of parametric motions in a highly efficient manner. Within

Furcifer this mechanism was used specifically to generate continuously ascending and descending tones that appear prior, during and after key thematic gestures in the work.

Table 6.2 Primary ai_trackWorld uses in Furcifer

Ai_TrackWorld	**Timing**	**Description**
	00:37-00:48	Ascending synthetic tones
	02:58-03:00	Descending swoop into key gesture
	08:30-08:40	Ascending tones culminating in the close of the work

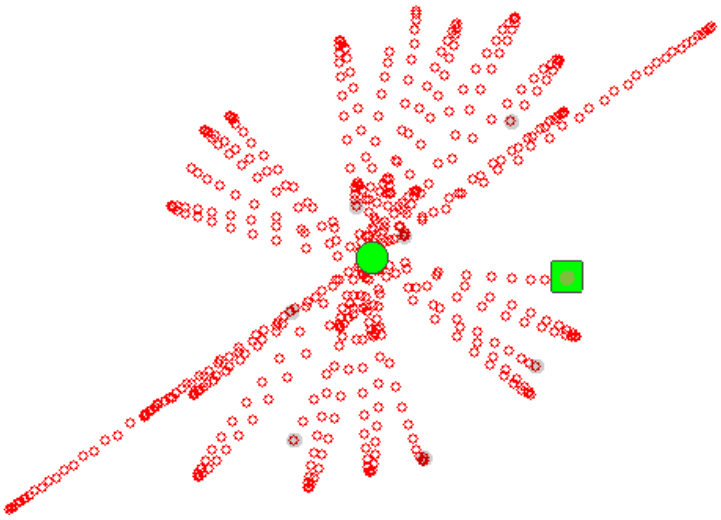

Figure 6.3 A graphical representation of a single ai_TrackWorld Agent moving towards the World Node as seen in the centre

```
function ai_TrackWorld()
  initai()
  changeMotionModel("Physics")
  local pos = getP("pos")
  local worldsize = getWorldSize()
  local props = {Mass = math.random(0.01,0.6),amp = 0.4,freq =pos:y()
      /261.625 ,pan =  ( (  pos:x() / worldsize:x())*2) -1 ,Attraction =
       2,dur = 5}
  addP(props)
  setP("Velocity",Vec2.new(math.random(-30,30),math.random(-30,30)))
  setP("SynthDef","default")
  while true do
    pos = getP("pos")
    setP("freq",pos:y()/261.625)
    setP("pan", ( (  pos:x() / worldsize:x())*2) -1)
    setP("amp",math.pow( getMeanCol(currentScreen()) /255,2 ))

    drawCircle(currentScreen(),pos,3,RGBA.new(255,0,0,150),-1)

    local targetPos = getPID("pos",0)
    local dir = (targetPos - pos):norm()
    setP("Thrust",dir* getP("Attraction"))
    coroutine.yield()
  end
  die()
end
```

Figure 6.4 ai_TrackWorld agent who continuously seeks the position of the World Node used in Furcifer

A second agent was designed to explicitly create high-frequency, pulse-based textures similar to those created by 'ai_HorizontalFollower' in *Modular Void* (see 0). 'ai_BasicFollower' was consequently devised to track and seek the closest other agent (see Figure 6.6 and Figure 6.5). This simple one-to-one tracking mechanism results in stable cyclic patterns when an agent is paired with another. When a number of other agents are near one another, the 'attraction' property which influences the thrust of the agents towards one another is highly significant. The default condition of this property highlights the nonlinear approach to the system and demonstrates the *butterfly effect* (Lorenz, 2000) in action. Small modifications to this variable can have radical consequences when a number of agents are in proximity to one another. In *Furcifer*, to ensure controllability, these agents were instantiated in pairs across the stereo field. A variety of cyclic beating patterns were subsequently created using this method and can be heard throughout section *A* (see Table 6.3).

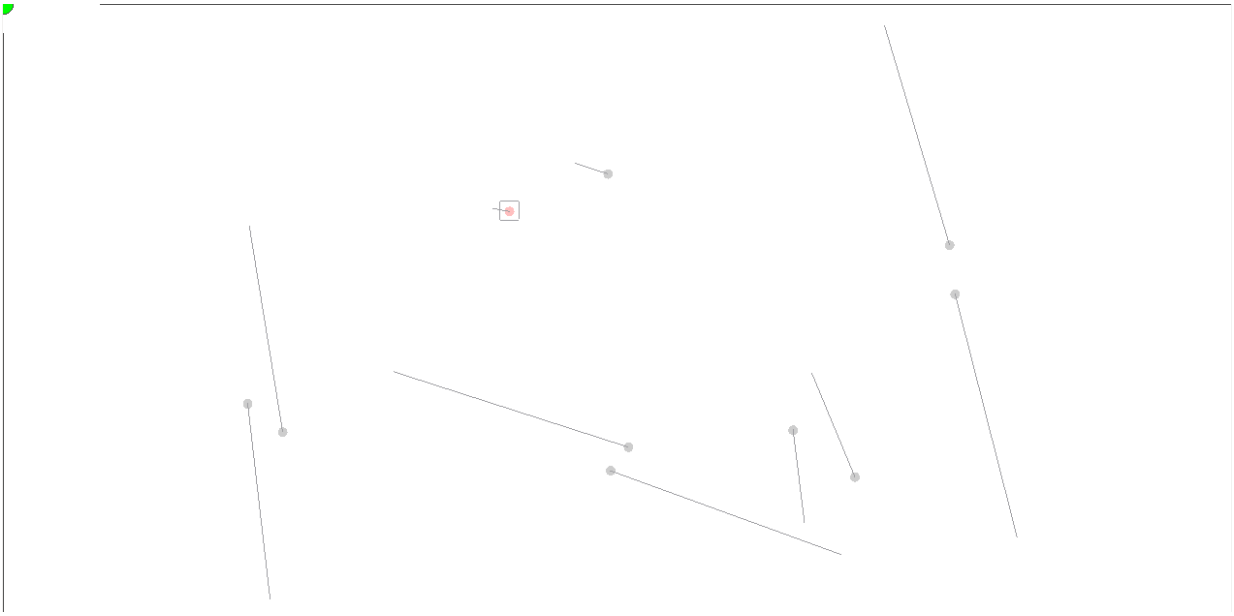

Figure 6.5 A graphical representation of the pairs of 'ai_BasicFollower' and their velocities

```
function ai_BasicFollower()
  initai()
  changeMotionModel("Physics")
  local pos = getP("pos")
  local worldsize = getWorldSize()
  local props = {Mass = math.random(0.01,0.6),amp = 0.4,freq =pos:y()
      /261.625 ,pan =  ( (  pos:x() / worldsize:x())*2) -1 ,attraction =
       1.2,dur = 20}
  addP(props)
  setP("Velocity",Vec2.new(math.random(-30,30),math.random(-30,30)))
  setP("SynthDef","default")

  while true do
    pos = getP("pos")
    setP("freq",pos:y()/261.625)
    setP("pan",  ( (  pos:x() / worldsize:x())*2) -1 )
    setP("amp",math.pow( getMeanCol(currentScreen()) /255,2 ))
    drawCircle(currentScreen(),pos,3,RGBA.new(255,0,0,150),-1)

    local targetId = getNearest(pos)
    local targetPos = getPID("pos",targetId)
    local dir = (targetPos - pos):norm()

    setP("Thrust",dir* getP("attraction"))
    coroutine.yield()
  end
  die()
```

Figure 6.6 The Basic Follower Agent used in Furcifer and Liten Röst

Table 6.3 ai_BasicFollower usage in Furcifer

Ai_BasicFollower	Timing
	00:31-00:40
	00:57-01:09
	01:21-01:30
	01:39-01:55
	04:35-04:53
	04:58-05:20

Agent-based image sonification played a key role in the creation of the main thematic gesture within the work (see Table 6.4). It was decided that this gesture would be constructed from a corpus of harmonically related source files, between which the granular synthesis agents would morph. As such the musical gesture was split across a number of voices and timbres in a fashion similar to Schoenberg's Klangfarbenmelodie[12] - a spectral synthesis of sorts. 'ai_ColourMangler' was consequently designed to generate grains using source files dictated by the brightness of a pixel (see Figure 6.7). The opening gesture and all subsequent variations throughout the composition were generated using this mechanism in conjunction with the path system. Each version of the gesture was subsequently generated through real-time variations in speed, and attraction to a single set path drawn over an image (see Figure 6.8).

Table 6.4 Key gesture and ai_ColourMangler usage in Furcifer

Ai_ColourMangler	Timing
	00:00-00:20
	02:38-02:50
	03:00-03:10
	06:32 – 07:17
	07:17 – 07:22
	07:57 – 08:19

[12] This form of synthesis is similar in design to that of CataRT by Ircam but is not based on an extensive corpus of descriptor analysed sounds (Schwartz et. al, 2006).

```
function ai_ColourMangler()                                                    1
  initai()                                                                     2
  local args = {range = 9,offset = 0,childdur = 0.2,rate = 0.01,x = 50,y       3
      = 50,amp = 0.1,rrate =  0.4,run = 1 }
  addP(args)                                                                   4
  local id = getID()                                                           5
  changeMotionModel("Physics")                                                 6
  while getP("run")>0 do                                                       7
    local colAvg = getMeanCol(currentScreen()) /255;                           8
    local args = {dur = getP("childdur"),amp = getP("amp"),bufNum = (          9
        colAvg * getP("range") )+getP("offset") }
    local x = getP("x")                                                        10
    local y = getP("y")                                                        11
    local child = spawnRelativeP(args,id,1,"ai_GrainA",(math.random()* x       12
        )- x*0.5,(math.random()* y)- y*0.5 )
    pause(getP("rate"))                                                        13
    setPID(args,child)                                                         14
  end                                                                          15
  die()                                                                        16
end                                                                            17
```

Figure 6.7 The colour mangler Agent script which used greyscale brightness to dictate the sound file to granulate

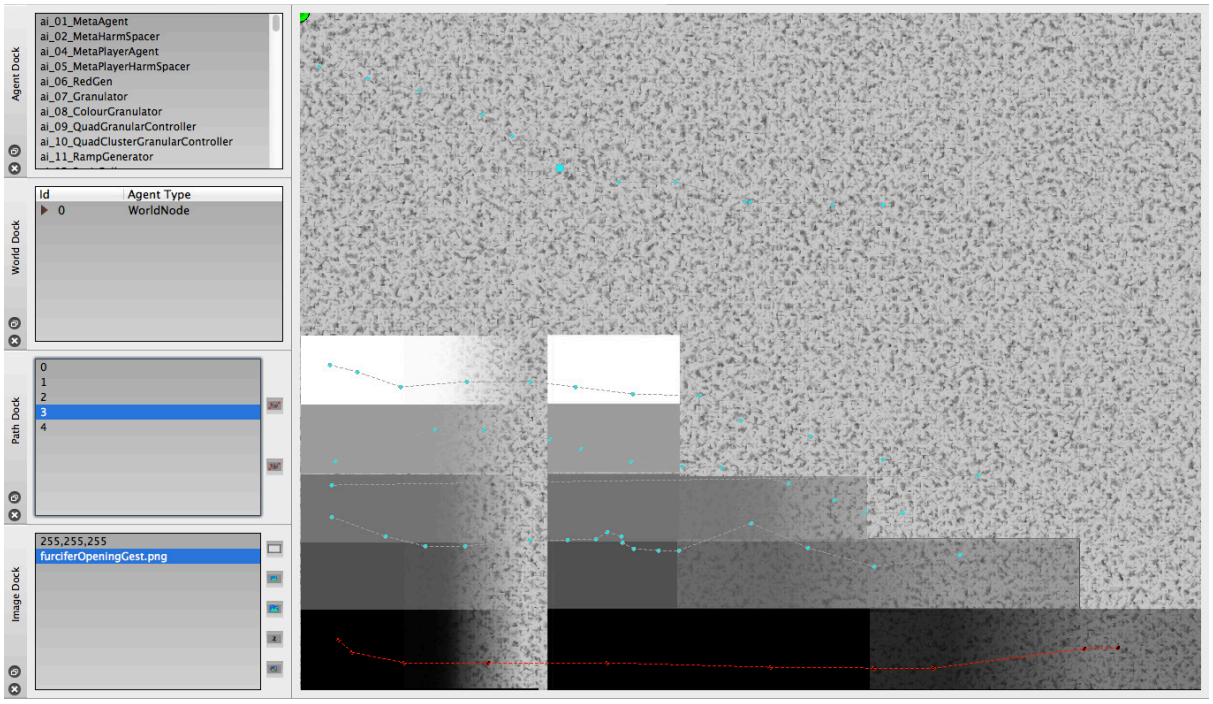

Figure 6.8 Example image used in conjunction with the path system to generate the main thematic gesture

Section *B* (03:00-06:31) presents contrasting material compared to surrounding sections. The sonic landscape mimics the depths of the ocean in a surreal fashion in which the listener is presented with a journey through a dark distal space before moving towards the unreal space in the reprise of section *A*. The section is underpinned by an agent who explicitly explores stigmergy - in which agents were indirectly influenced through the path system. 'ai_RedCulture' (see Figure 6.10) utilises imagery as an environment that it monitors *and* subsequently alters. This agent was designed to be attracted to high concentrations of red whilst seeking to avoid pixels with significant amounts of blue. Once the agent has finished computing where it should move to next, and has emitted its parametric data, it alters the pixel it is monitoring and those around it by drawing new data onto the surface. An agent designed to draw colours and gradients along a path was consequently constructed (see Figure 6.9) to set up the initial conditions for these agents. The drawing of paths and the stigmergetic interactions that follow in this instance behaves as a form of complex score generation. This approach is consequently similar to the *PheroMusic* system in which agents leave behind pheromone trails similar to that of ants and their colonies (Nymoen et

al., 2014). The open nature of agency in this instance allows these stigmergetic entities to vary over time and be altered in real-time. Pixel content in this environment behaves in a similar fashion to pheromones which other agents are consequently attracted to. The space becomes more environmental, the world more evolutionary and the music potentially more abstract. 'ai_RedCulture' is designed to alter the speed of playback, stereo pan position and amplitude of samples in the harmonic corpus with the horizontal position, vertical position and red pixel components respectively. This agent can be heard almost exclusively between 03:20-04:20 before other materials emerge alongside in the latter half of the section.

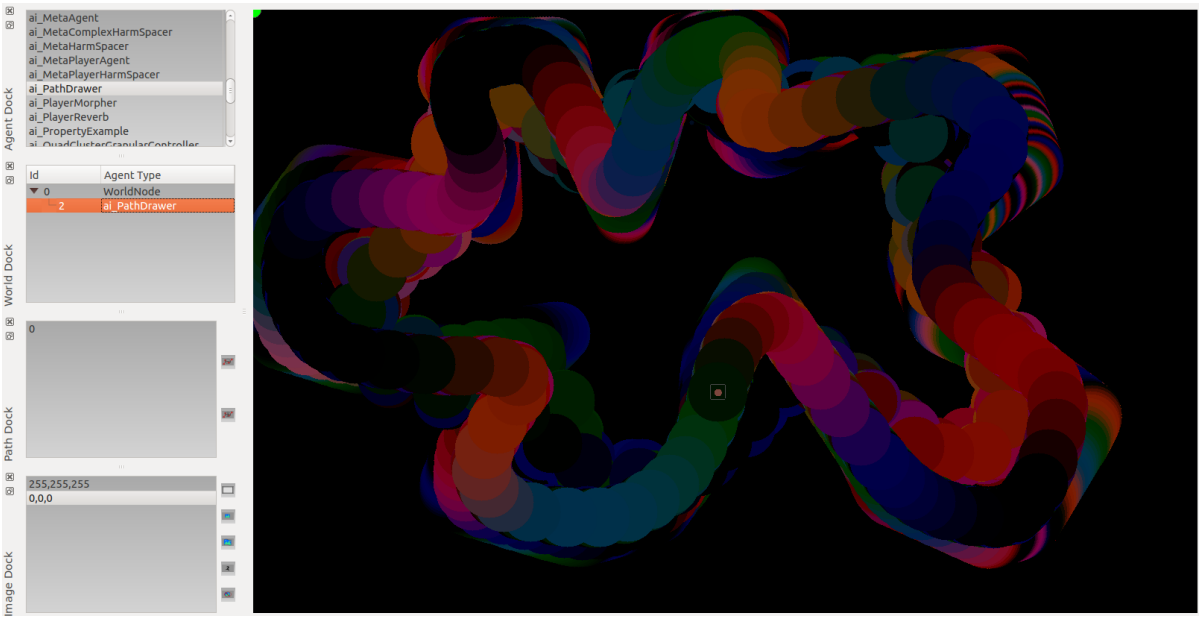

Figure 6.9 The output of an Agent which continuously draws on a user-defined path for other ai_RedCulture Agents to interact with

```
function ai_RedCulture()
  initai()
  local props = {rate = 1,amp = 0.1,pan = 0}
  addP(props)
  setP("SynthDef","player")
  changeMotionModel("Physics")
  setP("Thrust",Vec2.new(math.random(-10,10),math.random(-10,10)))
  setP("Velocity",Vec2.new(math.random(-10,10),math.random(-10,10)))
  while true do
    local pos =  getP("pos")
    local worldSize = getWorldSize()

    local col = getRGBA(currentScreen())
    setP("rate", pos:y()/261.625)
    setP("pan",((pos:x()/worldSize:x()) * 2)-1 )
    setP("amp", math.pow( col:r()/255,2) )
    setP("Thrust",getP("Thrust") * 0.33    )

    move_to_peak_red(pos,1.2)
    avoid_peak_blue(pos,0.5)

    if col:r() > 150 then
      spawnRelative(getParent(),1,"ai_RedCulture", math.random(-10,10),
          math.random(-10,10))
    end
    if col:b() > 180 then
  die()
    end
    drawCircle(currentScreen(),pos,5,col:r()* 0.5,col:g()* 0.75,col:b()
        *0.9,-1 )
    coroutine.yield()
  end
end
```

Figure 6.10 The Red Culture Agent used as part of Furcifer which is attracted to red content and is killed by blue whilst continuously transforming the underlying material

Section 6.03 Evaluation of the Relative Path System - *Furcifer*

In the traditional sense, paths within an n^{th} dimensional parametric space can be viewed as a means of automating controls over time. In allowing paths to influence agents, their impact on a controller or compositional system can be carefully regulated in real-time. When in use, agent movements within the parametric space can be altered to converge or dissipate from a repeatable path. This concept has a significant amount of artistic potential as a means of explicitly controlling parameters, or relinquishing control to the logic of an agent as part of a performance or composition. As the path system itself can be manipulated via the scripting system, agents could be devised to subsequently generate and manipulate paths in a prescriptive manner. As such, the path system itself can be a potential mediator of stigmergy.

Chapter 7. *Liten Röst* (2014) – 25:05

Liten Röst is a large-scale work in 5.1 format that originates primarily from a single short recording of a female voice. The work was completed in early 2014 at the Visby International Centre for Composers (VICC) in Gotland, Sweden and received its premiere at Sound Junction (2014) at the University of Sheffield. It was also performed at the 2014 International Computer Music Conference. Its title when translated into English means, 'Little Voice'. The work is 26:00 in duration, however, it can be split into four distinct movements. The first two of these (I - 09:30 and II - 05:50) can be performed as individual works. III and IV, despite their shorter duration, rely on materials that are exposed and developed in previous sections. Awareness of these previous instances, even if subconscious, is required to appreciate the concluding sections.

Unlike previous works, *Liten Röst* explores autonomous traversal of parametric space. The path and later stigmergic approaches to movement throughout a parametric space in the Agent Tool found in *Furcifer* were an effective means of controlling agent movement indirectly. In the case of *Liten Röst,* the work was used to explore abstracted and indirect movement through controlling emergent behaviours. The work explores two forms of agent-based tracking as a means of generating evolving musical textures and spatializing them around the listener. These are as follows:

1. Average group position tracking
2. Flocking simulations based on rules of separation, alignment and cohesion

Both of these approaches were consequently explored and evaluated throughout the compositional process using a number of different mapping techniques.

Throughout the compositional process and the development of appropriate agents for the work, experiments using colour data to invoke agents was conducted. While these experiments

were successful (see Section 7.01(a)), such techniques were not used as an active part of the compositional process until *Mikro Studie A* (see Section 8.01). In the course of conducting these experiments, live coding was used as a means of rapidly prototyping, developing, evaluating and refining agents and relationships.

Over the course of the work, sets of contrasting materials are used to create and explore an assortment of sonic spaces. While the majority of the work may appear to reside in distal space, forms of microsound, both natural and synthetic function as a means of transporting the listener back to a more direct proximate space (Smalley, 2007). Similar to the work *Furcifer* (see Chapter 6), the sonic landscapes that are present within *Liten Röst* morph between real, unreal and surreal spaces (Harrison, 1999). However, there are very few references to 'real' spaces in the work. Instead, *Liten Röst* utilises a variety of mimetic references to birds and beaches that are synthesised to invoke a warped view of "real objects in unreal spaces" (Wishart and Emmerson, 1996). These references were generated using the three tracking approaches mentioned previously and will be discussed at length in Section 7.01.

Liten Röst is composed in 5.1 surround sound, and much of the material is explored both panoramically and laterally around the listener. The energy of the spatial motions varies widely depending on the spectral content of the materials. In the context of this work, flocking algorithms are used as a means to spatialize high-frequency audio material. The dialogue between natural and synthetic microsound, when used in conjunction with these flocking algorithms, allows materials to traverse the continuum of Emmerson's *aural* and *mimetic* discourse (Emmerson, 2007). Furthermore, these materials traverse both perspective and distal space with position being used as a structural function throughout the work.

Liten Röst was envisaged as a large scale work over twenty-five minutes in duration and at the time of composition, this was the largest work created by the author. Careful attention was

paid to the perception of time and the structural form of the work when it was being composed. Across the work, materials are introduced to the listener gradually and are designed to emerge from spectromorphological content that is already present. Emmerson's axis of *abstract* and *abstracted* syntax was also of importance here (Emmerson, 1986a). A loose structural schema was created before commencing composition and evolved as materials were generated. Whilst the work at first may appear somewhat through-composed, each section draws upon and explores materials from surrounding sections to create relationships. Sections I and II both explore the purity of the female voice. Sustained notes and a limited number of harmonic overtones enabled subtle manipulations and variations of agents to be more identifiable when developing and exploring tracking mechanisms. These will be discussed in greater detail in Section 7.01(c)**.**

Across the first two sections, the perception of the female voice morphs between being very pure (real) to very synthetic invoking images of birds through the use of flocking algorithms. Throughout portions of *section I* however, forms of microsound using both the female voice and recordings of small shells (circa 02:45-04:20) are actively spatialized around the listener. Within this context, this material is underpinned by large vocal drones in the distal space. By juxtaposing these contrasting spaces, a clear dialogue emerges, with each acting as a continuous canopy designed to either attract or detract attention from the opposing material. In *section I* it is clear that the material existing within the distal space overpowers the more active microsound with a complete return to the distal space by approximately 05:00. *Section III* explores this more active microsound. *Section IV* is a reprise of *Section I* while emphasising the relationship between *I* and *III*.

Section 7.01 Agent Usage in *Liten Röst*

Liten Röst explores a collection of new agents designed specifically for the work, alongside a number of pre-existing agents. Additionally, a number of core mechanisms and agents from previous versions of The Agent Tool were refined. Agents and grains of microsound that are

influenced by artificial gravity were used to create a collection of glissandi. In a similar fashion to *Furcifer,* 'ai_TrackWorld' agents (see Section 6.02) generated these glissandi which are associated with the creation and release of tension (see Table 7.1).

Table 7.1 Ai_TrackWorld usage in Liten Röst

	Movement	**Timing**
ai_TrackWorld	2	00:00-00:15
	2	00:43-01:03
	2	05:22-05:58
	4	03:33-03:38
	4	04:26-04:30

The composition uses colour tracking agents to generate drone based materials in a similar fashion to those heard in *Modular Void* (see Section 5.03). 'Ai_ColourGranulator' (see Figure 5.17) was used to generate evolving drones to contrast with more active microsound. As with *Modular Void*, simple imagery generated pitch and panning variations in grains that underpin portions of the opening section (see Table 7.2). This technique is also prominent between 04:41 and 08:02. *Section III* differs from other sections in being gesture carried rather than texture carried. In the climax of this section, a bass pedal emerges between 02:04-03:15 which was devised using a live coded version of this agent[13].

Table 7.2 ai_ColourGranulator usage in Liten Röst

	Movement	**Timing**
ai_ColourGranulator	1	00:00-00:58
	1	01:57-02:16
	1	04:41-08:02
	3	02:04-03:15

[13]A discussion of live coding in the work can be found in 0

(a) Event Based Image Sonification

All forms of image sonification/audification in this research prior to *Liten Röst* were exclusively based on continuous parameter mappings. Gradually, a move towards repeatable gestures using granular synthesis was generated using a form of event-based sonification. This was achieved through the creation of a prototype agent which uses a set of purpose built Lua functions that tests for user specified RGBA configurations at the position of a given agent (see Figure 7.1 and Figure 7.2). This approach allows agents to be invoked based on specific colours within an image. This technique was used to generate a number of rhythmic pulses and gestures within *Liten Röst* (see Table 7.3). 'ai_ColourSpawner' was used to generate pulses of microsound that instantiate other instances (creating delayed variations) via imagery with strict bands of colour (see Figure 7.1, Figure 7.2 and Figure 7.3) between 02:46-04:33 in *section I* and the opening of *section III*. This 'barcode' technique became a key focus in *Mikro Studie A* in which imagery was used to dictate the timing of gestures within the work (see Section 8.01).

Table 7.3 ai_ColourSpawner usage in Liten Röst

	Movement	**Timing**
ai_ColourSpawner	1	02:46-04:33
	1	07:59-08:28
	2	00:25-00:40
	3	00:00-00:25

```
function colour_threshold( targetColour, threshold , id  )
  local col = getRGBAID(currentScreen(),id)
  if col:r() < (targetColour:r() - threshold) or col:r() > (targetColour
      :r() + threshold) then
    return false
  end
  if col:g() < (targetColour:g() - threshold) or col:g() > (targetColour
      :g() + threshold) then
    return false
  end
  if col:b() < (targetColour:b() - threshold) or col:b() > (targetColour
      :b() + threshold) then
    return false
  end
  return true
end

function colour_spawner(targetColour,threshold,id,agentName)
  local found = colour_threshold(targetColour,threshold,id)
  if found == true then
    spawnRelative(id,1,agentName,0,0)
  end
end
```

Figure 7.1 Lua functions created to detect and spawn agents based upon specified colours

```
function ai_ColourSpawner()
  initai()
  changeMotionModel("Physics")
  setP("Velocity",Vec2.new(25,0))
  setP("image",6)
  setP("dur",10)
  local id = getID()
  while true do
    colour_spawner(RGBA.new(255,0,0,120),20,id,"ai_GrainA")
    colour_spawner(RGBA.new(0,255,0,120),20,id,"ai_ColourGranulator")
    colour_spawner(RGBA.new(0,0,255,120),20,id,"ai_GravityGrain")
    coroutine.yield()
  end
  die()
end
```

Figure 7.2 The prototype colour spawner agent that spawned agents when approximate colours were detected

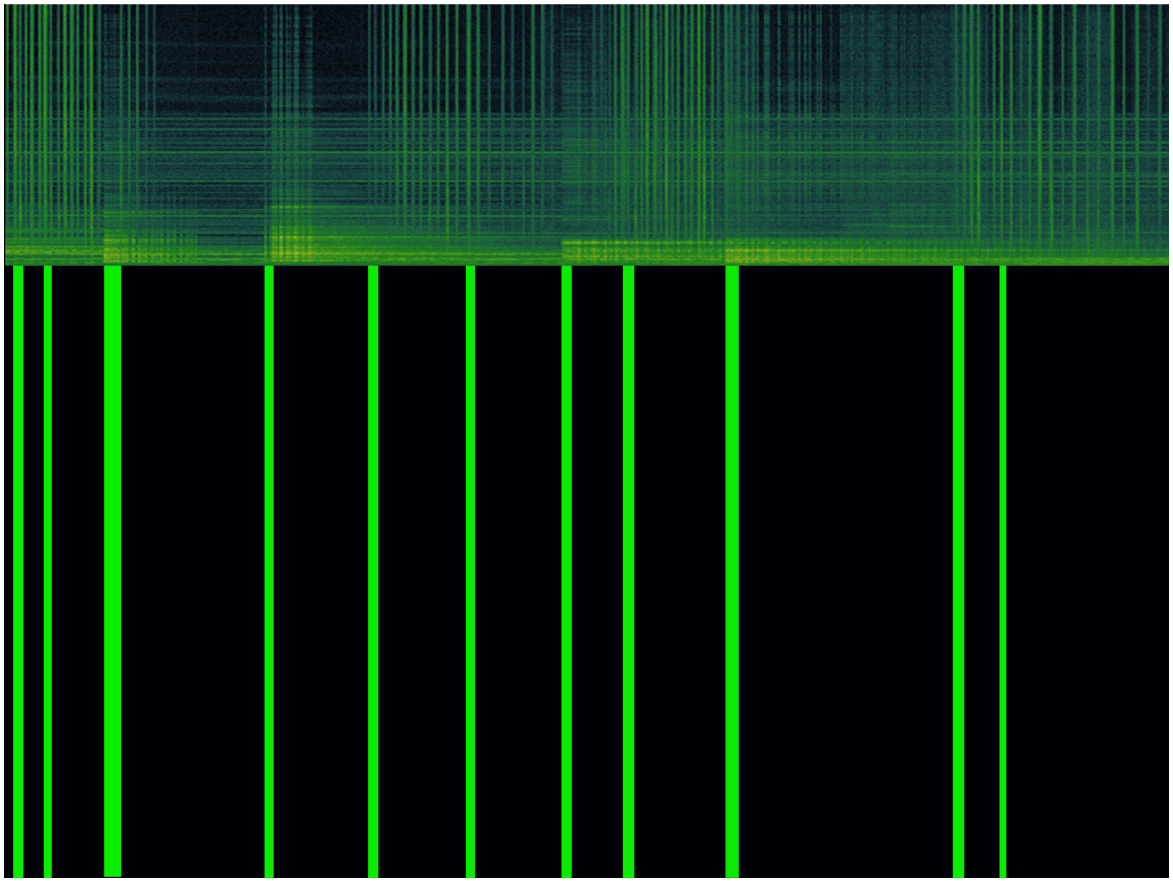

Figure 7.3 Barcode image used to generate pulses in Liten Röst Section I 02:46-02:59 alongside the spectrogram of the passage

(b) Live Coding in *Liten Röst*

Throughout the course of composing *Liten Röst* live coding was used as a means of experimenting and generating accompanying materials. Rather than being used as a performance technique it was used as a methodology for prototyping new agents in various sections in response to materials that had previously been written or generated. Two methodologies were used in the work to enable live coding of agents within the composition. The first of these was through the live coding of agents that are being continuously spawned by an existing control or generation mechanism. For example, the first instance of the boids flocking algorithm (see Section 7.01(c)) that is presented between 05:38-06:52 in *section I*, all child grains (which contain very high pitch material) that are spawned by the boid generators were altered live throughout the passage (mainly grain duration

and reverb). This performative technique (altering child agents) was subsequently used with the children of 'ai_ColourGranulator' throughout *section I* and *III* (see Table 7.2).

```
function ai_LiveSpawn()
  initai()
  addP("UpdateRate",0.25)
  local id = getID()
  local run = true;
  local count = 0
  while run do

    run = live_code(id,count) ---- live code callback function
    count = count +1
    pause(getP("UpdateRate"))
  end
  die()
end

--live code in this function..
function live_code(parentid,count)
  if count % 2 == 0 then
    spawnP({bufNum = 2 },parentid,1,"ai_GravityExample",100,100)
  end
  if count % 4 == 0 then
    spawnP({bufNum = 1 },parentid,1,"ai_GravityExample",100,400)
  end
  return true
end
```

Figure 7.4 The simple call-back mechanism used to live code agent spawning and control current instances

The second approach adopted as part of the composition was achieved via the creation of a callback function. If an agent in the system continuously invokes an arbitrary user-defined Lua function, upon saving the file containing the definition of that function[14], all instances of the agent in question will switch to using the newly compiled version of that function. This basic mechanism can be seen in 'ai_LiveSpawn' (see Figure 7.4) which was subsequently used to invoke boid agents,

[14] Upon saving a Lua file that has been loaded into the system, all functions and agents within that file will be recompiled.

and image sonification agents, such as 'ai_ColourGenerator', heard in their entirely throughout 02:25-04:44 in *section II*.

(c) Agent Tracking and Flocking in *Liten Röst*

Liten Röst further explores the impact of tracking agents within a parametric space. As previously mentioned, the work explores two new methods for performing agent tracking. Both of these methods are based on monitoring position but are distinguishable by the way in which this data is utilised. One of these is based upon simple logic. Each agent seeks an average position with those surrounding it. This rather basic approach allows agents to converge on a set position within the parametric space with a good degree of stability and predictability. 'ai_BasicGroupFollower' was created to generate a suite of materials in *Liten Röst* (see Figure 7.5 and Figure 7.8). This approach also allows the generation of simple flocks of agents within the parametric space. When a user influences the velocity of one member within a group, the velocity change subsequently alters the average position of a group which others are tracking. This indirectly allows the velocity variation to propagate through all agents within the group in a controllable fashion. It allows gravity to influence the velocity of the group throughout the space. Musically, this technique generated clusters of tones and grains and was employed within the final section of the work. 'ai_GroupGranulator' was designed to perform this function. This agent only tracks others of the same type whilst continuously spawning grains with pitch and position within the sound file mapped to the vertical and horizontal axes. Gravity was subsequently applied to the world that forced the agents to perform a pseudo- time stretch on given source files. This approach can be heard predominantly within the opening of the final section (see Table 7.4 and Figure 7.7).

Table 7.4 ai_GroupGranulator usage in Liten Röst

	Movement	**Timing**
ai_GroupGranulator	4	00:00:20
	4	00:24-00:58
	4	02:16-03:31

Two further agents were developed. 'ai_TrackWorldFlock' (see Figure 7.9) generates a flock of grains around the WorldNode. While the agent is functional it only allows the creation of one flock at a time. 'ai_PlayerReverb' (see Figure 7.10) plays a sound file through a reverb whose room size and wet/dry mix is influenced by the tracking algorithm. This agent was used to further process the closing materials of *section I* (08:40-09:15) - to metaphorically drag the materials into the distal space.

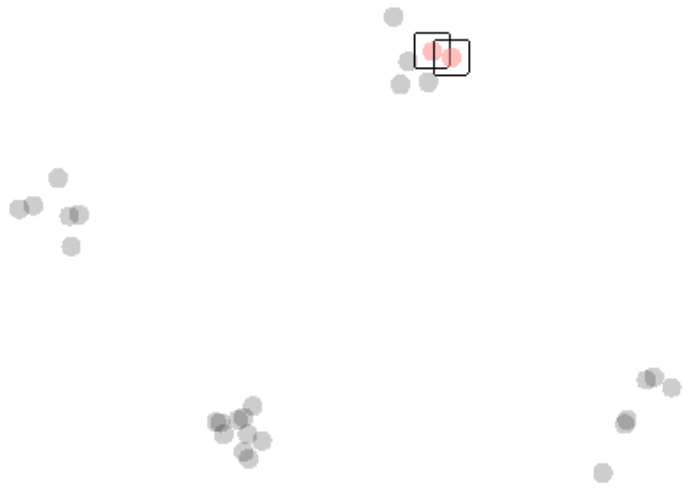

Figure 7.5 Several instances of ai_BasicGroupFollower flocking into small clusters at differing pitches across the stereo field.

```
function ai_BasicGroupFollower()
  initai()
  changeMotionModel("Physics")
  local pos = getP("pos")
  local worldsize = getWorldSize()
  local props = {Mass = math.random(0.1,0.6),amp = 0.4,freq =pos:y()
      /261.625 ,pan =  ( (  pos:x() / worldsize:x())*2) -1 ,attraction =
       1.2,range = 200,dur = 20}
  addP(props)
  setP("Velocity",Vec2.new(math.random(-30,30),math.random(-30,30)))
  setP("SynthDef","default")
  while true do
    pos = getP("pos")
    setP("freq",pos:y())
    setP("pan",  ( (  pos:x() / worldsize:x())*2) -1 )

    local nearby = getNearby(getP("range"))
    local avgPos = Vec2.new(0,0)
    local pos = getP("pos")
    for i = 1, #nearby do
      avgPos = avgPos + getPID("pos",nearby[i])
    end
    avgPos = avgPos/#nearby
    if avgPos:x() > 0 and avgPos:y() > 0 then
      local dir = (avgPos - pos):norm()
      setP("Thrust",dir* getP("attraction") * 0.5)
    end
    coroutine.yield()
  end
  die()
end
```

Figure 7.6 The code for the basic group follower used in early experiments in which agents are attracted to the average position of the surrounding agents.

```
function ai_GroupGranulator()
  initai()
  local id = getID()
  local props = {Mass = 0.3,rate = 0.01, childAgent = "ai_GrainB",
      xRandom = 3,yRandom = 3,childDur = 0.5,range = 300,attraction =
      4.3}
  addP(props)
  setP("Velocity",Vec2.new(20,0))
  changeMotionModel("Physics")
  while true do
    local xRandom = getP("xRandom")
    local yRandom = getP("yRandom")
    local x = math.random(xRandom* -1,xRandom) * 0.5
    local y = math.random(yRandom* -1,yRandom) * 0.5
    spawnRelativeP({dur = getP("childDur")},id,1,getP("childAgent"), x,y
        )
    move_to_average_type(getP("range"),getP("attraction"),"
        ai_GroupGranulator")
    pause( getP("rate"))
  end
  die()
end
```

Figure 7.7 ai_GroupGranulator agent who performs basic flocking and granular synthesis

Figure 7.8 Collections of group granulators creating textural drones in Liten Röst.

```
function ai_TrackWorldFlock()
  initai()
  setPID("pos",Vec2.new(400,400),0)
  local props = {FlockAttraction = 0.9,run = 1}
  addP(props)
  for i = 1, 10 , 1 do
    spawn(getID(),1,"ai_TrackWorld",math.random(0,1000),math.random
        (0,1000))
  end
  pause(1)
  while getP("run") > 0 do
    local attract = getP("FlockAttraction")
    print(numberOfChildren())
    for i = 0 , numberOfChildren()-1, 1 do
      setPID("Attraction",attract , getChild(i))
    end
    coroutine.yield()
  end
end
```

Figure 7.9 ai_TrackWorldFlock agent who spawns and controls ai_TrackWorld agents

```
function ai_PlayerReverb()
  initai()
  changeMotionModel("Physics")
  local pos = getP("pos")
  local worldsize = getWorldSize()
  local props = {Mass = math.random(0.01,0.6),amp = 0.4,mix =pos:y()/
      worldsize:y() ,room = pos:x() / worldsize:x() ,attraction = 1.2,
      dur = 20}
  addP(props)
  setP("Velocity",Vec2.new(math.random(-30,30),math.random(-30,30)))
  setP("SynthDef","reverbPlayer")
  while true do
    pos = getP("pos")
    setP("mix",pos:y()/worldsize:y())
    setP("room", pos:x() / worldsize:x() )
    local nearby = getNearby(getP("range"))
    local avgPos = Vec2.new(0,0)
    local pos = getP("pos")
    for i = 1, #nearby do
      avgPos = avgPos + getPID("pos",nearby[i])
    end
    avgPos = avgPos/#nearby
    if avgPos:x() > 0 and avgPos:y() > 0 then
      local dir = (avgPos - pos):norm()
      setP("Thrust",dir* getP("attraction") * 0.5)
    end
    coroutine.yield()
  end
  die()
end
```

Figure 7.10 ai_PlayerReverb agent that uses flocking dynamics to control reverb on sound files being played

An additional approach to tracking (used across all sections of the work) took the form of a reworking of Reynold's influential *boids* flocking algorithm (Reynolds, 1987). The 'boid_calculation' function was designed specifically to achieve this (see Figure 7.11). Similar to the original boids algorithm, each agent is aware of its neighbours within a given radius and the positions and velocities of each agent is governed by three rules. Firstly, agents must separate from those around it to avoid overcrowding. Secondly, agents are aligned to ensure that its velocity correlates with the average of all others within its flock. Finally, each agent attempts a form of cohesion by trying to move towards the mean position of the flock. The 'boid_calculation' function enhances this algorithm by allowing a user to specify what agent type is part of the same flock. This allows agents to flock explicitly with others of the same type or alternative entities. When agents utilise this mechanism to explore granular synthesis, the flocking simulation acts as an intuitive means of exploring microsound by traversing the content of a sound file in a natural and fluid fashion.

The agents 'ai_BoidGranulator' and 'ai_BoidFiveOneGranulator' (see Figure 7.12 and Figure 7.15) were both designed to enable the exploration of flocking granular synthesis in stereo and 5.1 surround. Both of these agents are control entities which spawn audible grains. 'ai_BoidGranulator' grains mapped pitch and sound file position to vertical and horizontal placements. 'ai_BoidFiveOneGranulator' child grains interacted with the position of speaker agents which were placed within the parametric space. The relative distance between the agent and each of these speakers consequently altered the grain's amplitude in each speaker. This subsequently allowed agents to flock around the listener in 5.1 surround sound[15]. 'ai_BoidFiveOneGranulator' was used to generate a suite of mimetic and non-mimetic materials throughout the first three sections. Within the *section I* it was used to generate flocks of abstract and noisy grains (02:44-05:03). These agents were then used to create mimetic materials imitating flocking birds by

[15] Whilst the materials here are presented in 5.1 surround, this mechanism was successfully adapted and tested in eight and sixteen channel surround sound with no perceivable latency.

granulating recordings of the female voice (05:38-06-52 and 08:30-09:28). In *section II* these agents were used to create highly synthetic flocks before using short noisy bursts in *section III* (see Table 7.5). This agent was used throughout the entirety of *section III* with multiple instances of these flocks being gradually presented alongside one another before building to the climax of the section at 03:15.

Table 7.5 ai_BoidFiveOneGranulator usage in Liten Röst

	Movement	**Timing**
ai_ BoidFiveOneGranulator	1	02:44-05:03
	1	05:38–06:52
	1	08:30–09:28
	2	01:05–02:45
	2	02:18–02:45
	3	00:48-01:38
	3	01:45-03:15
	3	03:22-03:47

```
function boid_calculation(position,range,cohesion,alignment,separation,
    type)
    local nearby = getNearby(range)

    local avgPos = Vec2.new(0,0)
    local align = Vec2.new(0,0)

    if type == not nil then
      for i = 1, #nearby do
        if getTypeID(nearby[i]) == type then

          avgPos = avgPos + getPID("pos",nearby[i])
          align = align +  getPID("Thrust",nearby[i])
        end
      end
    else
      for i = 1, #nearby do
        avgPos = avgPos + getPID("pos",nearby[i])
        align = align +  getPID("Thrust",nearby[i])
      end
    end
    avgPos = avgPos/#nearby
    align = align/#nearby

    local closest = getNearest(position)
    local closetPos = getPID("pos",closest)
    local closestDir = (closetPos - position):norm() * -1

    if avgPos:x() > 0 and avgPos:y() > 0 then
      local dir = (avgPos - position):norm()
      setP("Thrust",(dir * cohesion) + (align:norm()* alignment ) + (
          closestDir*separation ) )
    end
end
```

Figure 7.11 Boid_Calculation function used to impose flocking characteristics on agents in Liten Röst and Mikro Studie A

```
function ai_BoidGranulator()
  initai()
  local id = getID()
  local props = {Mass = 0.1,rate = 0.01, childAgent = "ai_GrainB",
      xRandom = 3,yRandom = 3,childDur = 0.5,range = 300,cohesion = 0.7,
       alignment = 1,separation = 0.1}
  addP(props)
  setP("Velocity",Vec2.new(20,0))
  changeMotionModel("Physics")
  while true do
    local xRandom = getP("xRandom")
    local yRandom = getP("yRandom")
    local x = math.random(xRandom* -1,xRandom) * 0.5
    local y = math.random(yRandom* -1,yRandom) * 0.5
    spawnRelativeP({dur = getP("childDur")},id,1,getP("childAgent"), x,y
        )
    boid_calculation(getP("pos"),getP("range"),getP("cohesion"),getP("
        alignment"),getP("separation"),ai_BoidGranulator)
    pause( getP("rate"))
  end
  die()
end
```

Figure 7.12 One of the boid granular synthesis agents used in Liten Röst wherein spawning agents flock around one another while continuously spawning grains

```
function ai_Speaker()
  initai()
  addP("speaker","left")
  while true do
    coroutine.yield()
  end
  die()
end

function ai_SpeakerFiveOneSpawner()
  initai()
  local left = spawn(0,1,"ai_Speaker",100,100)
  local right =spawn(0,1,"ai_Speaker",300,100)
  local centre = spawn(0,1,"ai_Speaker",200,75)
  local leftS = spawn(0,1,"ai_Speaker",100,300)
  local rightS = spawn(0,1,"ai_Speaker",300,300)
  pause(0.1)
  setPID("speaker","left",left)
  setPID("speaker","right",right)
  setPID("speaker","centre",centre)
  setPID("speaker","leftS",leftS)
  setPID("speaker","rightS",rightS)
  pause(1)
  die()
end
```

Figure 7.13 Agent mechanism utilised to set up speaker positions in the world

```
function calculate_five_one_coefficients()
 local numWorldChildren = numberOfChildrenID(0)
 local currentChildID = 0
 local speakerID = ""
 local delta = Vec2.new(0,0)
 local amp = 0
 for i = 0 , numWorldChildren-1, 1 do
   amp = 0
   currentChildID = getChildID(i,0)
   if getTypeID(currentChildID) == "ai_Speaker" then
    speakerID =  getPID("speaker",currentChildID)
    amp = (getPID("pos",currentChildID) - pos):mag() * 0.0001
    print(amp)
    if speakerID == "left" then
      setP("leftAmp",amp)
      elseif speakerID == "right" then
        setP("rightAmp",amp)
        elseif speakerID == "centre" then
          setP("centreAmp",amp)
          elseif speakerID == "leftS" then
            setP("leftSAmp",amp)
            elseif speakerID == "rightS" then
              setP("rightSAmp",amp)
            else
            end
          end
        end
      end
    end
  end
end
```

Figure 7.14 Agent based methodology used to calculate speaker amplitude coefficients

```
function ai_BoidFiveOneGranulator()
  initai()
  local id = getID()
  local props = {Mass = 0.1,rate = 0.01,range = 300,childAgent = "
      ai_GrainC",childDur = 0.5,xRandom = 3,yRandom = 3,childDur = 0.5,
      cohesion = 0.7, alignment = 1,separation = 0.1,freq = 440,leftAmp
      = 0,rightAmp = 0, centreAmp = 0, leftSAmp = 0, rightSAmp = 0}
  addP(props)
  setP("Velocity",Vec2.new(20,0))
  changeMotionModel("Physics")
  setP("SynthDef","fiveOneTester")
  print("boid 51 gran spawned")
  while true do
    local pos = getP("pos")
    local xRandom = getP("xRandom")
    local yRandom = getP("yRandom")
    local x = math.random(xRandom* -1,xRandom) * 0.5
    local y = math.random(yRandom* -1,yRandom) * 0.5
    spawnRelativeP({dur = getP("childDur")},id,1,getP("childAgent"), x,y
        )
    boid_calculation(getP("pos"),getP("range"),getP("cohesion"),getP("
        alignment"),getP("separation"),"ai_BoidFiveOneGranulator")
    coroutine.yield()
  end
end
```

Figure 7.15 ai_BoidFiveOneGranulator agent that performs five channel diffusion and granulation based on flocking dynamics

Section 7.02 Evaluating Agent Tool Flocking/Swarming

Flocking/Swarming provides a huge terrain for compositional and artistic exploration. The emergent properties of these systems have great appeal as means of traversing an abstract parametric space. While working with this flocking methodology, a number of mapping strategies and synthesis types were briefly explored. Directly mapping boids to amplitudes of sine waves created initial sound objects. However, such a direct approach to mapping and synthesis resulted in the dynamics of agents within collective clusters being hard to perceive; a viewpoint shared by Huepe et al. (2014). When working with forms of granular synthesis, these clusters quickly became interesting textural clouds. As the system can also spatialise real-time granular synthesis, it can be seen as a successful continuation of swarming systems such as Swarm Granulator (Blackwell and Young, 2004), BATBoids (Lewis, 2013), BMSwarmGranulator (Wilson, 2008) and the work by Huepe et al. (2014). The flocking system employed here is highly engaging to use due to the open nature of the scripting framework and the ability to code logic live. Consequently, an interesting area of future research is the live coding of a-life systems themselves.

From a compositional perspective, the open nature of the framework affords an appealing and somewhat organic method to traversing a parametric space. Flocking throughout the contents of a sound file and a physical performance space was an engaging and highly fruitful means of composing materials for *Liten Röst.* The explorations as part of this composition echoes two significant swarming strategies put forward by Schacher et al. (2011). Firstly, it is appropriate to map continuous changes in agent parameters to equally continuous changes in musical materials. Secondly, swams/flocks that have a tendency of settling into a stable configuration within the space may be best served controlling static materials within a work. The approach adopted in the context of *Liten Röst* adheres to the basic methodology that each agent is used to generate musical materials. In future, and in a fashion to that outlined by Schacher et al. (2011), these agents could also be utilised to create synergetic interactions. In this instance, agents could be utilised to draw

paths throughout an image with which other agent types (such as ai_RedCulture) could then interact. Another alternative approach worthy of investigation, after Bisig and Neukom (2008), would be the creation of proximity based events. When the properties of the agents or flock share common factors, such as a space or similar alignment values, musical events or agents could be invoked. This approach could also be utilised as a means of performing event-based image sonification wherein the flock is utilised to identify regions of interest within an image.

The ability to influence a flock in real-time is a highly desirable feature in any system. Besides the creation of control agents within the system, the Agent Tool also affords several other methodologies for higher-level control and interaction with flocking simulations. Each of these has a large amount of compositional research potential. Firstly the path system can be utilised to influence the flock in time. As the path system can be scripted and coded in real-time, algorithmic systems and compositions could be created. Furthermore, these systems can potentially be controlled via computer vision in a fashion similar to Unemi and Bisig (2004) but in an entirely open and scriptable manner.

Chapter 8. *Mikro Studie A* (2014) - 06:05

Mikro Studie A is a 5.1 work dedicated to exploring flocking and timbral variations of microsound generated through colour detection with predator-prey simulations being used as a basis for controlling granular synthesis. *Mikro Studie A* was completed in February 2014 at VICC shortly after the completion of *Liten Röst* (see Chapter 7) and many of the techniques and agents used in the former are present in this work. The composition is based on a refined usage of the Agent Tool to create a piece that contrasts previous work. *Mikro Studie A* is gesture carried with the majority of materials being heard in a close proximate space (Smalley, 2007). The microscopic grains that are generated using the Agent Tool are presented in what Smalley (2007) calls, *microphone space* in which the sonic images are amplified and then granulated directly to the listener with no added audio processing. As such the work aims to draw the listener's attention to the subtleties in the streams of grains and their motions in space around the listener. The composition reuses agents coded for previous versions of The Agent Tool. Each agent is provided a specific function within the *Mikro Studie A* and each will be discussed in turn.

The name of the work pays direct tribute to *Analogique A+B* by Iannis Xenakis for orchestra and tape which is considered to be the first composition to utilise a form of granular synthesis[16]1. Similar to *Analogique A+B*, *Mikro Studie A* is designed around a call and response between two contrasting sets of materials. *Analogique A* presents clouds of grains constructed from orchestral strings against *Analogique B* which presents synthetic sinusoidal material. The structure of the work is based on a continuous call and response between these two elements; the dialogue is held between two differing energy levels reinforced by the usage of pulse based materials.

Mikro Studie A draws inspiration from an assortment of differing approaches to granular synthesis. The work is driven by a constant sense of pulse and internal motion (motion within the

[16] The granular synthesis in Analogique B is constructed from sinusoidal grains that can be likened to Denis Gabor's construct known as 'Atoms' (Xenakis, 1992)

grains themselves). It was inspired by the presentation of streams of grains in *Points Critiques* and *Nodal* by Vaggione (2011) in which the speed of the underlying pulses is a key structural component. *Mikro Studie A* attempts to use this sense of pulse to draw the listener's attention to timbral variations and spatialization of the grains presented. As Vaggione (1994) suggests, analysis of events in microtime can be discussed with regards to syntax used in macrotime, without necessarily making the relationships between the two uniform (Vaggione, 1994, Vaggione, 1996). The structure of *Mikro Studie A* is based on a theme and variation, in which pulse based materials are used to focus the listener's attention to the intricacies of the grains, their spectral and timbral content and their spatial trajectories. Furthermore, while being primarily a synthetic work, source-bonded materials (primarily recordings of metallic objects) are used at key structural events for emphasis. Similar to *Points Critiques*, these materials are veiled in the background behind far more active materials.

The palette of sonic materials used within *Mikro Studie A* was deliberately constrained. This was done for two reasons, the first of which, is to create a consistent sound world/sonic landscape wherein all materials have very similar spectral content and energy. In doing so, the composition aims to encourage focused listening on the intricacies of the materials presented and their movement around the listener. Secondly, this decision was taken in the belief that explicit constraints sometimes allow creativity to flourish (Boden, 2004, Magnusson, 2010).

Unlike previous works, *Mikro Studie A* was composed with active diffusion in mind. In previous works such as *Liten Röst*, the textural nature of the work meant that rapid diffusion and motions imposed in real-time detracted from the content of the work. Being a gesture carried work, *Mikro Studie A* inherently affords greater possibilities for concert diffusion with gestures (on differing time-scales) and the release of energy from within acting as prime opportunities for a performer to alter the spatial positioning of such materials.

The usage of regular pulsed grains was similarly utilised as a means of aiding diffusion, suggesting positional changes take place on regular divisions of the pulses. Pulse-based grains emerge at several points in the piece, often utilising different pitch content and at different speeds. The pulses that follow the large gesture at 01:50 are, for example, similar to the materials found in the opening gesture, however, they are presented at different speeds. If in performance the diffuser somewhat replicates spatial trajectories at these points, the differences between the pulse-based materials will be highlighted, and the structure of the work will be further enhanced.

As *Mikro Studie A* is composed in 5.1 format, extensive spatial motions have been crafted as part of the compositional process and the extent to which a performer can actively diffuse the materials may be limited to the sound diffusion system being employed. Rather than diffusing the materials to specific locations (as is the case with the diffusion of stereo files), in performance, panoramic and lateral spaces presented within the work should simply be expanded and contracted. Furthermore, this means that at no point should any of the speakers in the core 5.1 set-up be reduced so that materials cannot be heard.

Section 8.01 Crafting *Mikro Studie A*

Mikro Studie A can be broadly split into two halves. Each of these halves explores differing approaches to agent-based granulation. The first half (00:00-03:04) was composed through explicit user control over a collection of agents. Throughout this section, a total of five different streams of audio are presented. The second half of the work (03:04 up to the closing gesture at 05:33) explores a purpose-built predator-prey system using agents where user-control is limited. This latter half uses the same source recording as many of the materials in the former, resulting in the section acting as a complex development of previous materials.

The first half of the composition is underpinned by several closely linked musical gestures. These gestures were constructed through an exploration of event-based colour sonification. Each

of these gestures comprises several agents performing at once but is signified by glass hits. The timing for these hits was directly generated from imagery that was crafted as part of the composition. Images comprising vertical bands of colour akin to a barcode were created to be read by an assortment of agents (see Figure 8.2). 'ai_RedGenerator' (see Figure 8.1) was utilised to generate all five key gestures in the opening half of the work (see Table 8.1). Rather than using the colour matching functions explored briefly in *Furcifer* (see Section 6.02), the agent was designed to monitor the amount of red at a given position. This approach was highly efficient and resulted in the generated barcodes acting as basic repeatable scores.

Table 8.1 Gestures created by ai_RedGenerator in Mikro Studie A

	Gesture Timing
Ai_RedGenerator	00:00
	00:14
	00:48
	01:49
	02:56

To generate materials that increased and decreased in amplitude either side of these gestures in time, the barcode patterns that were used were blurred with gradients applied, increasing and decreasing the red around these bars (see Figure 8.3). The basic 'ai_Granulator' (see Figure 5.16) was utilised with grains whose amplitudes were dictated by pixel brightness, to generate drones that crescendo and diminuendo around the key thematic gestures.

```
function ai_RedGenerator()                                                    1
  initai()                                                                    2
  changeMotionModel("Physics")                                                3
  local props = {AgentToSpawn = "ai_GravityGrain", spawnRate = 0.1,spawn      4
      = 1, threshold = 150}
  addP("AgentToSpawn","ai_GravityGrain")                                      5
  setP("Velocity",Vec2.new(10,0))                                             6
  setP("image",6)                                                             7
  setP("dur",30)                                                              8
  while getP("spawn") == true do                                              9
    local pos =  getP("pos")                                                  10
    if getRedFromPoint(currentScreen(),pos) > getP("threshold") then          11
     spawnRelative(getID(),1,getP("AgentToSpawn"),0,0)                        12
     pause(getP("spawnRate"))                                                 13
   end                                                                        14
   coroutine.yield()                                                          15
 end                                                                          16
end                                                                           17
```

Figure 8.1 The red generator used to spawn agents in the Mikro Studie A

The opening half of the work utilises two other agents with defined roles. 'ai_ColourGranulator' (see Figure 5.17), previously used in *Furcifer* and *Liten Röst*, was utilised to generate slowly evolving low-frequency drones that provide a backdrop for the contrasting microsound and gestures above them. In the first two instances, this agent was used to provide a backdrop for the flocking algorithms that were being explored in the first half of the composition. The agent then appears alongside the predator-prey simulation for the entirety of the second half acting as a bridge. The agent was also utilised to generate the repetitive pulses which underpin the opening half of the work.

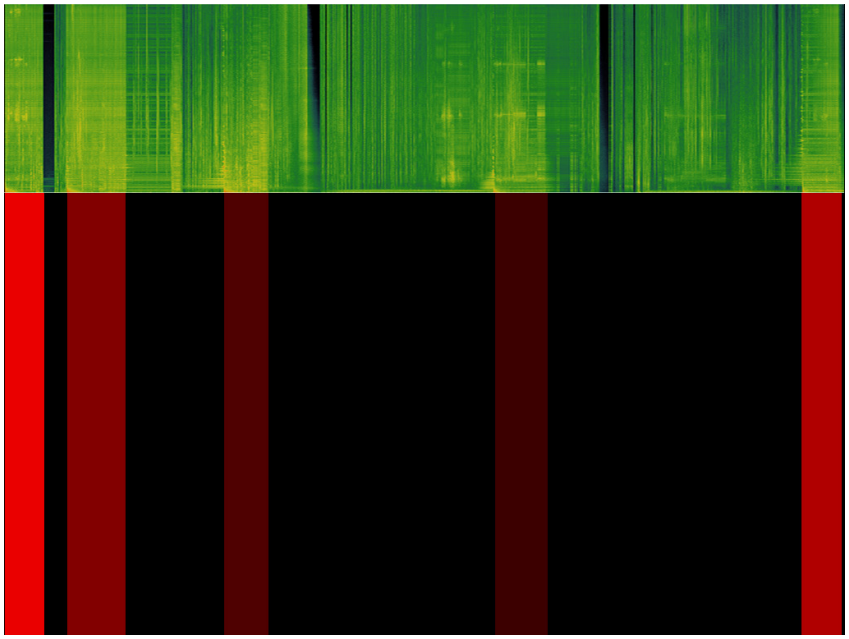

Figure 8.2 The red barcodes used to generate primary events in Mikro Studie A 00:00-03:04

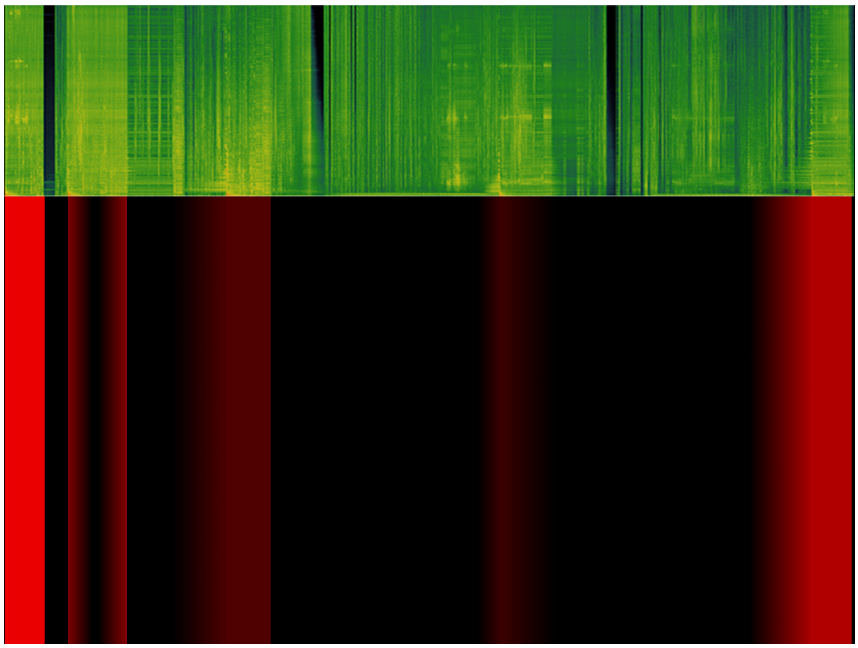

Figure 8.3 Red gradients used to generate swells in Mikro Studie A 00:00-03:04

Table 8.2 ai_ColourGranulator usage in Mikro Studie A

Ai_ColourGranulator	Timing
	01:13-01:49
	02:22-02:55
	03:06-04:54

The opening half of the composition also uses 'ai_BoidFiveOneGranulator' (also previously used in *Liten Röst* (see Figure 7.15). Unlike the audible flocking trajectories that can be heard in *Liten Röst,* the flocks used in *Mikro Studie A* were purposefully less identifiable. As such, the cohesion and alignment of the flocks were deliberately very low.

Table 8.3 ai_BoidFiveOneGranulator usage in Mikro Studie A

Ai_ BoidFiveOneGranulator	Timing
	00:03-0:08
	00:14-00:27
	00:38-00:48
	00:49-1:06
	01:10—1:38
	02:20-02:41

Section 8.02 Predator-Prey Simulations

The second half of *Mikro Studie A* was generated almost exclusively through a predator-prey simulation (see

Table 8.4) . This simulation algorithmically spawns and kills granular synthesis agents with a small collection of controls to influence the likelihood of generating offspring for either predator or prey at regular intervals. This was achieved through the core tracking functions within the system and property interface. Allowing agents to kill one another when they are within proximity can be conceived as a simple `predator' model as there is no differentiation between agent types. The property system was utilised to store a property that represents an agents 'type'. Subsequently,

when a predator monitors the world around it, it can check that an agent is of the right type before seeking towards it to destroy it. This mechanism underpins both the inaudible 'ai_Predator' and audible 'ai_MikroPreyGrainSynth' (see Figure 8.4 and Figure 8.5). The raw prey agent has no means of instantiating other prey. Instead, it continuously invokes child grain agents until it is killed. When used in conjunction with the predator, a predictable and linear sequence of events emerge. Collections of sounds are spawned, and the predators hunt these sounds until they cease to exist. To overcome this, a higher-level control agent, 'ai_MikroLifeSystem' (see Figure 8.6) was created to spawn predators and potential prey simultaneously while providing higher levels of user control over these agents. Most notably, this agent controls the percentage outcome of predator or prey to be spawned at the next given time interval. These probabilities are independent of one another and can be controlled in real-time allowing the simulation to be controlled and biased. To stop the predators from simply spawning and killing agents indefinitely, predators were given a limit to the number of prey they could kill before they themselves died. This limit was consequently controlled in real-time when the simulation was invoked. This mechanism was actively used to replicate the pulsing grains of the first half of the composition between 05:01-05:34. In this passage, the thrust for the prey was set extremely high with a low kill limit. Spawning these at regular intervals consequently resulted in prey being killed almost immediately after they had been spawned. The thrust control of the predator became a key musical control as it directly dictated the lifetime of all audible materials in the simulation.

Table 8.4 ai_MikroLifeSystem usage in Mikro Studie A

Ai_ MikroLifeSystem	**Timing**
	03:04-04:56
	03:47-04:56
	05:01-05:34

```
function ai_Predator()                                                    1
        initai()                                                          2
        changeMotionModel("Physics")                                      3
        setP("Velocity",Vec2.new(math.random(-150,150),math.random        4
            (-150,150)  ))
        setP("dur",10)                                                    5
        local kills = 0                                                   6
        local run = true                                                  7
        while run do                                                      8
   if kills > 3 then                                                      9
    run = false                                                           10
    break                                                                 11
  end                                                                     12
  local nearby = getNearby(500)                                           13
  local closestPrey = 0                                                   14
  local diff = 1000                                                       15
  local found =false                                                      16
  local currentDiff =1001                                                 17
  for i = 1 , #nearby-1 do                                                18
    if hasPID("creature",nearby[i]) then                                  19
      found = true                                                        20
      currentDiff =(getPID("pos",nearby[i]) - getP("pos"):mag()           21
      if currentDiff < diff then                                          22
        diff = currentDiff                                                23
        closestPrey = nearby[i]                                           24
      end                                                                 25
    end                                                                   26
  end                                                                     27
  if found == true then                                                   28
   local targetPos = getPID("pos",closestPrey)                            29
   local dir = (targetPos -pos ):norm()                                   30
   setP("Thrust",dir)                                                     31
   if dir:mag() < 1 then                                                  32
     killID(closestPrey)                                                  33
     kills = kills + 1                                                    34
   end                                                                    35
  end                                                                     36
  die()                                                                   37
end                                                                       38
```

Figure 8.4 ai_Predator agent designed to track and hunt prey agents

```
function ai_MikroPreyGrainSynth( )                                    1
        initai()                                                       2
        local props = {creature = 0,rate = 0.01,cdur = 0.5,xDeviation = 3
            2, yDeviation=2,childType = "ai_GrainB",run = 1,cbufNum = 0,
             cAmp = 0.7,lowerR = 0.01,upperR =0.05,bufRange = 0,bufNum =
             0}
        local id = getID()                                             4
        changeMotionModel("Physics")                                   5
        addP(props)                                                    6
        setP("Velocity",Vec2.new( math.random(-100,100),math.random    7
            (-100,100)  ))
        while getPID("run",id) > 0 do                                  8
                local col = getMeanCol(0)                              9
                local cArgs = {dur = getPID("cdur",id), bufNum = getPID( 10
                    "bufNum",id) + (col/255)*getPID("bufRange",id),
                    creature = 0  }
                local xDev = getPID("xDeviation",id)                   11
                local yDev = getPID("yDeviation",id)                   12
                local child = spawnRelativeP(cArgs,id,1,getPID("       13
                    childType",id), math.random(xDev* -1.0,xDev), math.
                    random(yDev* -1.0,yDev)  )
                pause(  getPID("rate",id) + ((math.random(0,1000)/1000)* 14
                    getPID("upperR",id)) + getPID("lowerR",id) )
        end                                                            15
        pause(10)                                                      16
        die()                                                          17
end                                                                    18
```

Figure 8.5 ai_MikroPreyGrainSynth agent used to spawn grains that are prey

```
function ai_MikroLifeSystem()                                          1
        initai()                                                       2
        local props = {  rate = math.random(0,1000)/1000 * 3 ,         3
            grainThreshold = 90,predatorTreshold = 30,noPredatorSpawn =
            5,childBuffer = 0}
        local id = getID()                                             4
        addP(props)                                                    5
        while true do                                                  6
                if math.random(0,100) <  getPID("grainThreshold",id)   7
                    then
                        local child = spawnRelative(id,1,"             8
                            ai_MikroPreyGrainSynth",math.random
                            (-100,100),math.random(-100,100))
                        coroutine.yield()                              9
                        coroutine.yield()                              10
                        setPID("bufNum",getPID("childBuffer",id),child) 11
                                                                       12
                end                                                    13
                if math.random(0,100) <  getPID("predatorTreshold",id) 14
                    then
                        for i = 1, getPID("noPredatorSpawn",id), 1 do  15
                                spawnRelative(id,1,"ai_Predator",math. 16
                                    random(-100,100),math.random
                                    (-100,100))
                        end                                            17
                end                                                    18
                pause(getP("rate"))                                    19
        end                                                            20
        die()                                                          21
end                                                                    22
```

Figure 8.6 The predator-prey system that was used in Mikro Studie A

Section 8.03 Evaluating the Agent Tool Post *Mikro Studie A*

Mikro Studie A suggested that event-based image sonification was possible through the use of abstract agents. The system employed here was an effective tool for generating musical events and gestures. The use of a single colour channel in this instance resulted in image creation becoming an engaging means generating events, as events could easily and accurately be predicted when designing the images to be used. Similar to Metasynth, the creation of musical events was driven both by the agents employed and the drawings they used. While the scripting interface can be used to detect specific colours within an image, perceiving when and where these events will be invoked within a complex image is problematic. Images with diverse colour content would require extensive user experimentation and training to be able to learn and accurately predict how events will be generated. This approach to RGB detection enables the entire 8 bit RGB colour palette to be used to invoke events or unique synthesis voices. In practice, a smaller colour pallette of up to 32 unique colours could be used to create a predictable sonic output. As such, imagery could be used as a means of generating event-based image sonifications more akin to audification. Alternatively, the scripting framework could be utilised to perform blob or edge detection algorithms as a means of identifying objects within an image with a relatively small processing overhead.

The successful experimentation and usage of predator-prey systems within *Mikro Studie A* demonstrated that complex granular synthesis algorithms could be used effectively in an organic fashion. *Mikro Studie A* also led to the conclusion that complex predator-prey relationships and simulations have great potential for further investigation. The open nature of the scripting framework including the property and path systems means that complex and bespoke control mechanisms could easily be employed. When using predator-prey models in such a context, allowing the composer to tweak, balance and re-balance the simulation plays a major role in the composition process. It is this interaction that allows users to participate actively in the simulation rather than being mere observers who set up initial environmental conditions.

Chapter 9. Conclusions

Across the course of the portfolio of compositions, organic and evolutionary texture design became more and more refined. As the Agent Tool's control mechanisms became more defined so did the development of musical textures and their corresponding agents. Familiarity and continual usage of certain mapping strategies led to proficiency in agent definition and control culminating in the agents used in *Liten Röst* and *Mikro Studie A*. During the composition of earlier works, the feature set and the scripting interface for the Agent Tool was continuously being developed. When work commenced on *Liten Röst*, the feature set of the scripting interface became standardised. Consequently, these two final works are the most refined and focused, with new techniques being explored explicitly within the scripting engine itself. While developing a familiarity with the scripting interface, the focus of compositional development moved towards the creation of evolving textures. All of the compositions in the portfolio except *Mikro Studie A* were subsequently texture-carried (Smalley, 1986). *Mikro Studie A* marked a departure in the compositional style in being primarily gesture-carried. The scriptable predator-prey ecosystem that was designed for the work enabled controllable and predictable gestures to be created as part of the compositional process. This foray into real-time control over predator-prey systems as a means of generating gestural materials was largely successful and could provide the basis for future compositional research. In the future, it is possible that entire compositions could be scored through the control of such algorithms.

Furthermore, conclusions from this portfolio suggest that agency can effectively be used as a means of designing sonic gestures and textures. The Agent Tool was used predominantly to create texture-carried compositions and the interface and agent paradigm enabled slowly evolving textures and drones to appear with relative ease. The impact of an agent's surroundings and an agent's relationships to others are often not perceivable until relevant data or controls are tweaked. Real- time control over these agent systems subsequently requires time to develop once

spawned. The approach to agency explored here consequently favours real-time control and exploration of sustained textures. The separation of control/spawning agents with audible grains, when combined with the motion models of the software, offer a highly controllable and perceivable influence over the internal motion of grain-based textures. Altering the gravity of child grains or attraction/repulsion to paths allows them to converge or diverge from the current point in the parametric space. This approach to creating internal motion was successfully used across *Furcifer* and *Liten Röst.* The flexibility of the motion system successfully enables the creation of predictable and repeatable motions throughout the parameter space if required. As seen in *Furcifer* this can be used as a means of creating defined gestures. This methodology is somewhat time-consuming and less immediate than exploring the creation of textures as agents need to be invoked, monitored, and destroyed before altering the path or motion logic within the agent. This process has the potential to be significantly more time consuming because of the varied nature of both parametric mappings and sound sources. To create explicit and concise gestures with agency in this manner users must be familiar with both the content of the source sound file and the parametric mappings employed by the agent/s being invoked.

The compositions and the software suggest that agency and ecosystemic approaches to exploring parameter spaces and granular synthesis algorithms are worthy of further creative investigation. The open nature of the scripting framework and the feature set provided means that the development of new controllers and algorithms is possible for those with little programming experience. This is important if the software is ever to be used by others. Initial exploration needs to be quick and it needs to generate reasonable results from basic presets and tutorials. Most importantly, it needs to motivate the composer to learn more and become familiar with the concepts involved. The evolutionary nature of the spawn protocols (flocking, predator/prey) naturally encourages engagement.

The design of the software removes many requirements for users to know complex programming concepts[17]. As the framework itself was designed for a broad range of uses with the potential for highly complex simulations that utilise either large numbers of calculations or agents, the refresh rate of the system was deliberately restricted. By default, the refresh rate of the system is locked to 60 hertz. The framework is consequently inappropriate to audio rate audifcations. This refresh rate directly represents the lowest possible latency in the system. The speed of Lua execution also plays a crucial role. Due to the micro-threaded nature of Lua, once an agent has finished performing a task at a given speed it must yield to allow other agents' threads to run. It is, therefore, possible for a heavy computational load in one agent to hold up and offset the execution of another agent if it does not yield appropriately.

Conclusions from the portfolio also indicate that imagery is an effective modality for stimulating granular synthesis through the intuitive use of agent-based ecosystems. For events to be predictable, it became apparent that imagery with clear and straightforward colour relationships should be used. Across the works, images with bands of colour and gradients were successfully used to automate parametric controls predictably. Imagery with rapid changes in pixel content, on the other hand, can be hard to predict but has the potential to generate interesting automation patterns. The software is therefore highly successful in performing continuous parametric sonification. As agents can freely traverse imagery, the system also behaves as an effective model-based sonification environment. Imagery has the potential to set out parametric controls which the user can then excite, control and subsequently perform if they so wish. The experiments in event-based sonification culminating in *Mikro Studie A,* indicated that simple imagery and colour can be used to invoke musical events and gestures. When using raw pixel data

[17] The scripting API ensures that users do not have to worry about different data types and defining their own class that inherits the properties of another. The inheritance model is the most commonplace approach to designing agent-based systems and innately relies upon the user to be aware of classes and inheritance in either Java or C++.

to invoke events, complex image data make the sequence of events hard to predict. Blob detection (Danker and Rosenfeld, 1981) or object detection algorithms could be utilised to trigger events within the system. However, the trade-off here is an overhead within the scripting interface resulting in a slowdown in real-time execution. If this slowdown could be overcome, traversing/influencing an ecosystem or invoking events with real world objects is a highly appealing area for future research.

Across the creation of this portfolio, the open approach to designing multi-agent systems allowed agents to be easily repurposed and adapted for different compositional ideas. The approach to agency used here continuously offered new avenues for exploring musical creativity. The LUA interface allowed the creation of agents and control structures that were highly engaging and easy to adapt. The concept of agency works both as a simple function (performing a finite action) and as a complex evolutionary process where not only can granulations occur but a granulation system can spawn further granulations (the sonic material becoming both audio and control data). With such a vast scope of possibilities, careful constraints should be designed when using such an open system. In this context, imagery was utilised to constrain explorations successfully in conjunction with a human bottleneck regulating agent development. The simple scripting interface that the system offers enabled the evolutionary approach of refining agents to be largely enjoyable and efficient with new agents generating exciting controls and materials.

Bibliography

Ash, K. M. & Stavropoulos, N. (2011). *Livecell: Real-time score generation through interactive generative composition*, Ann Arbor, MI: Michigan Publishing, University of Michigan Library.

Assayag, G. (2014). Creative Symbolic Interaction. 40th Intl. Comp. Mus. Conf. and 11th Sound and Music Comp. Conf.(ICMC/SMC joint conf.). ICMA, SMC, National and Kapodistrian University of Athens, IRMA, pp 1-6.

Ben-Tal, O., Berger, J., Cook, B., Daniels, M. & Scavone, G. (2002). SonART: The sonification application research toolbox. Presented at the 8th International Conference on Auditory Display (ICAD) Kyoto, Japan

Biles, J. (1994). GenJam: A genetic algorithm for generating jazz solos. Proceedings of the International Computer Music Conference. INTERNATIONAL COMPUTER MUSIC ACCOCIATION, 131-131.

Biles, J. A. (1998). Interactive GenJam: Integrating real-time performance with a genetic algorithm. Proceedings of the 1998 international computer music conference. 232-235.

Biles, J. A. (2007). Evolutionary computation for musical tasks. *Evolutionary computer music.* Springer.

Biles, J. a. & Eign, W. G. (1995). GenJam Populi: Training an IGA via audience-mediated performance. *San Francisco, USA*, 347-348.

Bilotta, E., Miranda, E. R., Pantano, P. & Todd, P. M. (2002). Artificial life models for musical applications: Workshop report. *Artificial life,* 8**,** 83-86.

Bisig, D. & Neukom, M. (2008). Swarm based computer music-towards a repertory of strategies. *computer music,* 12**,** 13.

Blackwell, T. & Young, M. (2004). Self-organised music. *Organised Sound,* 9**,** 123-136.

Boden, M. A. (2004). *The creative mind: Myths and mechanisms*, Psychology Press.

Bruel, P. & Queiroz, M. (2014). *A Protocol for creating Multiagent Systems in Ensemble with Pure Data*.

Burtner, M. (2006). Perturbation Techniques for Multi-Agent and Multi-Performer Interactive Musical Interfaces. Proceedings of the New Interfaces for Musical Expression Conference (NIME 2006). 4-8.

Burton, A. R. & Vladimirova, T. (1999). Generation of musical sequences with genetic techniques. *Computer Music Journal,* 23**,** 59-73.

Cariani, P. (1991). Emergence and artificial life. *Artificial life II,* 10**,** 775-798.

Cariani, P. (1997). Emergence of new signal-primitives in neural systems. *Intellectica,* 2**,** 95-143.

Chadabe, J. (1997). *Electric sound : the past and promise of electronic music,* Upper Saddle River, N.J., Prentice Hall.

Dahan, K. (2005). *Domaines formels et representations dans la composition et l'analyse des musiques electroacoustiques.* Ph. D. thesis, CICM, Université Paris VIII, France.

Dahlstedt, P. (2001a). Creating and exploring huge parameter spaces: Interactive evolution as a tool for sound generation. Proceedings of the 2001 International Computer Music Conference. 235-242.

Dahlstedt, P. (2001b). Creating and exploring the huge space called sound: interactive evolution as a composition tool. Proc. of Music without Walls, Music without Instruments Conf.

Dahlstedt, P. (2001c). A MutaSynth in parameter space: interactive composition through evolution. *Organised Sound,* 6**,** 121-124.

Dahlstedt, P. (2007). Evolution in creative sound design. *Evolutionary computer music.* Springer.

Dahlstedt, P. (2009). Thoughts on creative evolution: A meta-generative approach to composition. *Contemporary Music Review,* 28**,** 43-55.

Dahlstedt, P. & Mcburney, P. (2006). Musical agents: toward computer-aided music composition using autonomous software agents. *Leonardo,* 39**,** 469-470.

Danker, A. J. & Rosenfeld, A. (1981). Blob detection by relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence***,** 79-92.

Dartnall, T. (2002). *Creativity, cognition, and knowledge: An interaction*, Greenwood Publishing Group.

Darwin, C. (1897). The origin of species by means of natural selection, or, The preservation of favored races in the struggle for life. Vol. 1. *International Science Library*.

Dawkins, R. (1986). *The blind watchmaker: Why the evidence of evolution reveals a universe without design*, WW Norton & Company.

Dawson, B. & Entertainment, H. (2001). Micro-Threads for Game Object Al. *Game Programming Gems 2***,** 258.

De Campo, A. (2007). Toward a sonification design space map. Proc. Int Conf. on Auditory Display (ICAD), Montreal, Canada. International Community for Auditory Display.

Dhomont, F. (1996). Is there a Québec sound? *Organised Sound,* 1**,** 23-28.

Dixon, S. (2000). A lightweight multi-agent musical beat tracking system. Pacific Rim International Conference on Artificial Intelligence. Springer, 778-788.

Eigenfeldt, A. (2007). The creation of evolutionary rhythms within a multi-agent networked drum ensemble. *ICMC, Copenhagen, Denmark*.

Eigenfeldt, A. (2008). Multi-agent Modeling of Complex Rhythmic Interactions in Real-time Performance. *Sounds of Artificial Life: Breeding Music with Digital Biology*.

Eigenfeldt, A. (2010). Coming together: negotiated content by multi-agents. Proceedings of the 18th ACM international conference on Multimedia. ACM, 1583-1586.

Eigenfeldt, A. & Pasquier, P. (2011). A sonic eco-system of self-organising musical agents. European Conference on the Applications of Evolutionary Computation. Springer, 283-292.

Eldridge, A. (2005). Cyborg dancing: generative systems for man machine musical improvisation. *Proceedings of Third Iteration*.

Emmerson, S. (1986a). *The Language of electroacoustic music,* London, Macmillan.

Emmerson, S. (1986b). The relation of language to materials. *The language of electroacoustic music.* Springer.

Emmerson, S. (2007). Music, Electronic Media and Culture. Farnham: Ashgate Publishing Ltd,.

Flowers, J. H. (2005). Thirteen years of reflection on auditory graphing: Promises, pitfalls, and potential new directions. *Faculty Publications, Department of Psychology***,** 430.

Flowers, J. H., Buhman, D. C. & Turnage, K. D. (1997). Cross-modal equivalence of visual and auditory scatterplots for exploring bivariate data samples. *Human Factors: The Journal of the Human Factors and Ergonomics Society,* 39**,** 341-351.

Garcia, R. (2001). Growing sound synthesizers using evolutionary methods. Proceedings ALMMA 2001: Artificial Life Models for Musical Applications Workshop,(ECAL 2001).

Gartland-Jones, A. & Copley, P. (2003). The suitability of genetic algorithms for musical composition. *Contemporary Music Review,* 22**,** 43-55.

Gimenes, M., Miranda, E. R. & Johnson, C. (2006). The development of musical styles in a society of software agents. Proceedings of the International Conference on Music Perception and Cognition. 4.

Goto, M. (2001). An audio-based real-time beat tracking system for music with or without drum-sounds. *Journal of New Music Research,* 30**,** 159-171.

Goto, M. & Muraoka, Y. (1996). Beat tracking based on multiple-agent architecture-a real-time beat tracking system for audio signals. Proceedings of the Second International Conference on Multiagent Systems. 103-110.

Grond, F. & Berger, J. (2011). Parameter mapping sonification. *The sonification handbook***,** 363-397.

Harrison, J. (1999). Imaginary Space-Spaces in the Imagination. Australasian Computer Music Conference.

Hermann, T. (2008). Taxonomy and definitions for sonification and auditory display. Proceedings of the 14th International Conference on Auditory Display (ICAD2008) Paris, France.

Hermann, T. & Hunt, A. (2004). The importance of interaction in sonification. *In:* BARRASS, S. A. V., P., ed. Proceedings of ICAD 04. Tenth Meeting of the International Conference on Auditory Display, Sydney, Australia. International Community for Auditory Display.

Hermann, T. & Hunt, A. (2005). An introduction to interactive sonification. *IEEE multimedia***,** 20-24.

Hodgson, P. (1999). Modelling cognition in musical improvisation through evolution. Proceedings of the AISB'99 Symposium on Musical Creativity. 15-19.

Hodgson, P. (2002). Artificial evolution, music and methodology. Proceedings of the 7th International Conference on Music Perception and Cognition. 244-247.

Hogg, B. & Vickers, P. (2006). Sonification abstraite/sonification concrete: An'aesthetic persepctive space'for classifying auditory displays in the ars musica domain. *In:* TONY STOCKMAN, L. V. N., CHRISTOPHER FRAUENBERGER, ALISTAIR D. N. EDWARDS AND DEREK BROCK, ed. Proceedings of the 12th International Conference on Auditory Display (ICAD) London, UK. International Community for Auditory Display.

Holland, J. H. (1975). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*, U Michigan Press.

Horner, A. & Goldberg, D. E. (1991). Genetic Algorithms and Computer-Assisted Music Composition. Proceedings of the 1991 International Conference on Genetic Algorithms and Their Applications. 337.

Huepe, C., Colasso, M. & Cádiz, R. F. (2014). Generating music from flocking dynamics. *Controls and Art.* Springer.

Husbands, P., Copley, P., Eldridge, A. & Mandelis, J. (2007). An introduction to evolutionary computing for musicians. *Evolutionary computer music.* Springer.

Ierusalimschy, R., De Figueiredo, L. H. & Celes Filho, W. (1996). Lua-an extensible extension language. *Softw., Pract. Exper.,* 26**,** 635-652.

Jacob, B. (1995). Composing with genetic algorithms. International Computer Music Conference (ICMC '95) Banff Alberta. 452-455.

Johnson, C. G. (1999). Exploring the sound-space of synthesis algorithms using interactive genetic algorithms. Proceedings of the AISB Workshop on Artificial Intelligence and Musical Creativity, Edinburgh. Citeseer.

Kaliakatsos–Papakostas, M., Floros, a., Drossos, K., Koukoudis, K., Kyzalas, M. & Kalantzis, A. (2014). *Swarm Lake: A Game of Swarm Intelligence, Human Interaction and Collaborative Music Composition*, Ann Arbor, MI: Michigan Publishing, University of Michigan Library.

Kramer, G. (1993). *Auditory display: Sonification, audification, and auditory interfaces*, Perseus Publishing.

Kramer, G., Walker, B., Bonebright, T., Cook, P., Flowers, J. H., Miner, N. & Neuhoff, J. (2010). Sonification report: Status of the field and research agenda.

Lewis, A. (2013). *Bangor Audio Toolkit* [Online]. Available: http://www.bangor.ac.uk/music/studios/BAT/. [Accessed 09/08 2016].

Lohner, H. (1986). the UPIC system: A User's Report. *Computer Music Journal,* 10**,** 42-49.

Lorenz, E. (2000). 7. The Butterfly Effect. *The chaos avant-garde: Memories of the early days of chaos theory,* 39**,** 91.

Magnusson, T. (2010). Designing constraints: Composing and performing with digital musical systems. *Computer Music Journal,* 34**,** 62-73.

Malt, M. (2004). Khorwa: A Musical Experience with Autonomous Agents. Proceedings of International Computer Music Conference. International Computer Music Association, 59-62.

Mandelis, J. (2001). Genophone: An evolutionary approach to sound synthesis and performance. *Proceedings ALMMA***,** 37-50.

Mandelis, J. (2002). Adaptive hyperinstruments: applying evolutionary techniques to sound synthesis and performance. Proceedings of the 2002 conference on New interfaces for musical expression. National University of Singapore, 1-2.

Mandelis, J. & Husbands, P. (2003). Musical interaction with artificial life forms: Sound synthesis and performance mappings. *Contemporary Music Review,* 22**,** 69-77.

Manning, P. (2012). The oramics machine: From vision to reality. *Organised Sound,* 17**,** 137-147.

Manning, P. (2013). *Electronic and computer music*, Oxford University Press.

Marino, G., Serra, M.-H. & Raczinski, J.-M. (1993). The UPIC system: Origins and innovations. *Perspectives of New Music,* 31**,** 258-269.

Martins, J. & Miranda, E. R. (2008). Engineering the Role of Social Pressure: A New Artificial Life Approach to Software for Generative Music. *Journal on Software Engineering,* 2**,** 31-42.

McCormack, J. (2001). Eden: An evolutionary sonic ecosystem. European Conference on Artificial Life. Springer, 133-142.

McCormack, J. (2003). Evolving sonic ecosystems. *Kybernetes,* 32**,** 184-202.

McCormack, J. (2012). Creative ecosystems. *Computers and creativity.* Springer.

Meyers, S. (2005). *Effective C++: 55 specific ways to improve your programs and designs*, Pearson Education.

Minsky, M. (1988). *Society of mind*, Simon and Schuster.

Miranda, E. (2001a). *Composing music with computers*, CRC Press.

Miranda, E. R. (1995). Granular synthesis of sounds by means of a cellular automaton. *Leonardo***,** 297-300.

Miranda, E. R. (2000). The art of rendering sounds from emergent behaviour: Cellular automata granular synthesis. Euromicro Conference, 2000. Proceedings of the 26th. IEEE, 350-355.

Miranda, E. R. (2001b). Evolving cellular automata music: From sound synthesis to composition. Proceedings of 2001 Workshop on Artificial Life Models for Musical Applications.

Miranda, E. R. (2003). On the music of emergent behavior: what can evolutionary computation bring to the musician? *Leonardo,* 36**,** 55-59.

Moore, A. M., D (2010). USSS Toolkit. https://www.sheffield.ac.uk/usss.

Murray-Rust, D. (2008). Musical Acts and Musical Agents: theory, implementation and practice.

Murray-Rust, D. & Smaill, A. (2005). Musical acts and musical agents. Proceedings of the 5th Open Workshop of MUSICNETWORK.

Murray-Rust, D., Smaill, A. & Edwards, M. (2006). MAMA: An architecture for interactive musical agents. *Frontiers in Artificial Intelligence and Applications,* 141**,** 36.

Murray-Rust, D., Smaill, A. & Maya, M. C. (2005). VirtuaLatin-towards a musical multi-agent system. Sixth International Conference on Computational Intelligence and Multimedia Applications (ICCIMA'05). IEEE, 17-22.

Nymoen, K., Chandra, A., Glette, K. H., Tørresen, J., Voldsund, A. & Jensenius, A. R. (2014). PheroMusic: Navigating a Musical Space for Active Music Experiences.

Paetsch, F., Eberlein, A. & Maurer, F. (2003). Requirements Engineering and Agile Software Development. WETICE. 308.

Paine, G. (2002). Interactivity, where to from here? *Organised Sound,* 7**,** 295-304.

Pearse, S. & Moore, D. (2014). An agent based approach to interaction and composition. Proceedings of the International Computer Music Conference. International Computer Music Association, 810-814.

Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. *ACM SIGGRAPH computer graphics,* 21**,** 25-34.

Rhea, T. L. (1972). *The evolution of electronic musical instruments in the United States.* George Peabody College for Teachers.

Roads, C. (1996). *The computer music tutorial,* Cambridge, Mass., MIT Press.

Roads, C. (2001). *Microsound,* Cambridge, Mass., MIT Press.

Rouzic, M. (2008). PhotoSounder. http://photosounder.com/.

Scaletti, C. (1994). Sound synthesis algorithms for auditory data representations. SANTA FE INSTITUTE STUDIES IN THE SCIENCES OF COMPLEXITY-PROCEEDINGS VOLUME-. ADDISON-WESLEY PUBLISHING CO, 223-223.

Schacher, J., Bisig, D. & Neukom, M. (2011). Composing with swarm algorithms—creating interactive audio-visual pieces using flocking behaviour. Proceedings of the International Computer Music Conference. 100-107.

Schwarz, D., Beller, G., Verbrugghe, B. & Britton, S. (2006). Real-time corpus-based concatenative synthesis with catart. 9th International Conference on Digital Audio Effects (DAFx). 279-282.

Sedes, A., Courribet, B. & Thiébaut, J.-B. (2004). Visualization of sound as a control interface. Proceedings of the Digital Audio Effects Conference, Naples.

Sims, K. (1991). Artificial evolution for computer graphics. SIGGRAPH '91 Proceedings of the 18th annual conference on Computer graphics and interactive techniques. ACM, 319-328.

Smalley, D. (1986). Spectro-morphology and structuring processes. *The language of electroacoustic music.* Springer.

Smalley, D. (1997). Spectromorphology: explaining sound-shapes. *Organised sound,* 2**,** 107-126.

Smalley, D. (2007). Space-form and the acousmatic image. *Organised sound,* 12**,** 35-58.

Spicer, M. (2004). AALIVENET: an agent based distributed interactive composition environment. International Computer Music Conference. 1-6.

Spicer, M. (2014). A Multi-agent Interactive Composing System for Creating “expressive” Accompaniment. Proceedings of the International Computer Music Conference Athens, Greece. Ann Arbor, MI: Michigan Publishing, University of Michigan Library.

Spicer, M., Tan, B. & Tan, C. (2003). A Learning Agent Based Interactive Performance System. Proceedings of the International Computer Music Conference. 95-98.

Stavropolous, N. (2005). *Portfolio of Compositions.* PhD thesis, University of Sheffield.

Stavropolous, N. (2006). Multi-channel formats in electroacoustic composition: Acoustic space as a carrier of musical structure. Proceedings of Sound and Music Computing Conference, Lefkada, Greece. 251-254.

Thiebaut, J.-B., Healey, P. G., Kinns, N. B. & Mary, Q. (2008). Drawing electroacoustic music. Proc. ICMC.

Todd, S. & Latham, W. (1992). Evolutionary art and computers London: Academic Press.

Ueda, L. K. & Kon, F. (2004). Andante: Composition and performance with mobile musical agents. In Proceedings of the International Computer Music Conference 2004. Citeseer.

Unemi, T. & Bisig, D. (2004). Playing music by conducting BOID agents-A style of interaction in the life with A-Life. *Proceedings of A-Life IX***,** 546-550.

Vaggione, H. (1994). Timbre as syntax: a spectral modeling approach. *Contemporary Music Review,* 10**,** 73-83.

Vaggione, H. (1996). Articulating microtime. *Computer Music Journal,* 20**,** 33-38.

Vaggione, H. (2001). Some ontological remarks about music composition processes. *Computer Music Journal,* 25**,** 54-61.

Walker, B. N. & Nees, M. A. (2011). Theory of Sonification. *In:* NEUHOFF, J. G. (ed.) *The Sonification Handbook.* Berlin: Logos Verlag.

Waschka II, R. (2007). Composing with genetic algorithms: GenDash. *Evolutionary Computer Music.* Springer.

Wenger, E. (2015). MetaSynth. http://www.uisoftware.com/MetaSynth/index.php: U&I Software.

Wessel, D. L. (1979). Timbre space as a musical control structure. *Computer music journal***,** 45-52.

Whalley, I. (2009). Software Agents in Music and Sound Art Research/Creative Work: current state and a possible direction. *Organised Sound,* 14**,** 156-167.

Wilson, S. (2008). Spatial swarm granulation. Proceedings of the 2008 International Computer Music Conference. SARC, ICMA.

Wishart, T. & Emmerson, S. (1996). *On sonic art*, Psychology Press.

Wooldridge, M. & Jennings, N. R. (1995). Intelligent agents: Theory and practice. *The knowledge engineering review,* 10**,** 115-152.

Worrall, D. (2010). Parameter mapping sonic articulation and the perceiving body. *Proceedings of the 16th International Conference on Auditory Display (ICAD2010).* Washington DC, USA: International Community for Auditory Display.

Wulfhorst, R. D., Flores, L. V., Nakayama, L., Flores, C. D., Alvares, L. O. C. & Viccari, R. (2001). An open architecture for a musical multi-agent system. Proceedings of Brazilian symposium on computer music.

Wulfhorst, R. D., Nakayama, L. & Vicari, R. M. (2003). A multiagent approach for musical interactive systems. Proceedings of the second international joint conference on Autonomous agents and multiagent systems. ACM, 584-591.

Xenakis, I. (1992). *Formalized music: thought and mathematics in composition*, Pendragon Press.

Yeo, W. S. & Berger, J. (2005a). Application of image sonification methods to music. *Online document*.

Yeo, W. S. & Berger, J. (2005b). A framework for designing image sonification methods. Proceedings of International Conference on Auditory Display Limerick, Ireland. International Community for Auditory Display.

Yeo, W. S. & Berger, J. (2006). Application of raster scanning method to image sonification, sound visualization, sound analysis and synthesis. Proceedings of the International Conference on Digital Audio Effects. Citeseer, 309-314.

Yeo, W. S., Berger, J. & Lee, Z. (2004). SonART: A framework for data sonification, visualization and networked multimedia applications. Proceedings of the 2004 International Computer Music Conference. 180-184.

Young, G. (1989). *The Sackbut Blues: Hugh Le Caine, Pioneer in Electronic Music*, National Museum of Science and Technology.

Discography

Adkins, M. (2009). Five Panels. Compact Disc. Signature/Radio France.

Bayle, F. (1992). Erosphère. Compact Disc. Ina-GRM. INA-C 3002.

Bernier, N. (2012). Travaux Mécaniques. Compact Disc. empreintes DIGITALes. IMED 12114.

Bouchard, C. (2004). Fractures. Compact Disc. empreintes DIGITALes. IMED 0474.

Dhomont, F. (1996). Forêt Profonde. Compact Disc. empreintes DIGITALes. IMED 9634.

Gobeil, G. (2001). ...dans le silence de la nuit... Compact Disc. empreintes DIGITALes. IMED 0155.

Gough, H. (2008). With What Remains. Compact Disc. Entr`acte. E38.

Gough, H. (2010). Mikrolimata. Compact Disc. Entr`acte. E91.

Harrison, J. (1996). Articles Indéfinis. Compact Disc. empreintes DIGITALes. IMED 9627.

Harrison, J. (2007). Environs. Compact Disc. empreintes DIGITALes. IMED 0788.

Lewis, A. (2007). Mirois Obscurs. Compact Disc. empreintes DIGITALes. IMED 0789.

Moore, A. (2011). Contrechamps. Compact Disc. empreintes DIGITALes. IMED 11112.

Stollery, P. (2011). Scènes. Compact Disc. empreintes DIGITALes. IMED 11111.

Thigpen, B. (2011). Divide By Zero. Compact Disc. Sub Rosa. SR 317.

Vaggione, H. (2011). Point Critiques. Compact Disc. Ina-GRM. INA-G 6032.
Wishart, T. (1992). Red Bird/Anticredos. Compact Disc. Emf Media. B0000560TL.
Young, J. (2007). Lieu-Temps. Compact Disc. empreintes DIGITALes. IMED 0787.
Zanesi, C. (1996). Arkeion. Compact Disc. Ina-GRM. INA-E 5001.
Zanèsi, C. (1998). Le Paradoxe de la Femme-Poisson. Compact Disc. Ina-GRM. INA-K 198.

Appendix: Selected Performances

Overture (2012)

iFIMPaC 2013 (International Festival for Innovations in Music Produc- tion & Composition, Leeds, UK)

NYCEMF 2013 (New York Contemporary Electroacoustic Music Festival

Pompeu Fabra University 2013 (Barcelona, Spain)

Modular Void Suite (2013)

iFIMPaC 2014 (International Festival for Innovations in Music Produc- tion & Composition, Leeds, UK)

Furcifer (2013)

From Tape to Typedef: Compositional Methodologies in Electroacoustic Music 2013 (Sheffield, UK)

Liten Röst (2014)

ICMC 2014 (International Computer Music Conference, Athens, Greece)

Sound Junction 2014 (Sheffield 2014)

Mikro Studie A (2014)

Sound Junction 2014 (Sheffield 2014)

iFIMPaC 2015 (International Festival for Innovations in Music Production & Composition, Leeds, UK)

Sonorities 2015 (Belfast, UK)