The Audience of the Singular

Cale Plut

B.F.A. Simon Fraser University, 2015

Project submitted in partial fulfillment of the
Requirements for the degree of
Master of Fine Arts

in the
School for Contemporary Arts
Faculty of Communication, Art, and Technology

Simon Fraser University
Summer 2017

Approval

Name:	**Cale Plut**
Degree:	**Master of Fine Arts**
Title:	**The Audience of the Singular**

Examining Committee:

Chair: Eldritch Priest
Assistant Professor

Arne Eignfeldt
Senior Supervisor
Professor

Philippe Pasquier
Supervisor
Associate Professor
School of Interactive Arts and Technology

Andrew Brown
External Supervisor
Professor
Griffith University

Date Defended/Approved: May 26, 2017

Abstract

The Audience of the Singular explores and challenges the traditional role of audiences. *The Audience of the Singular* uses an interactive video game form to allow the audience to interact with the performance in real-time. Each time the game is played, new music is generated by a corpus-based, machine-learning system. The audience then collaborates with an arranger system and a virtual audience to play through their own personal musical performance. By utilizing the interactive, real-time nature of the video game, *The Audience of the Singular* removes the audience from its traditional role as a passive spectator, and instead places them in an active authorship role in their own unique performance.

Keywords: Generative Music; Games; Audience; Interaction

Table of Contents

Approval……..ii
Abstract……...iii
Table of Contents……iv
List of Figures……..……v

Defence Statement…….....…….……1
Works Cited……...……..13

Appendix A: Cyborg Collaborators…….....……..15
Works Cited……..……..33

Appendix B: Quarter Circle Forwards …….....……35

Appendix C: PureData Patch…….....…….40

Appendix D: Manual……...…….42

List of Figures

Figure 1	Gameplay Screenshot	...9
Figure 2	Analysis flowchart	16
Figure 3	Generative flowchart	..18
Figure 4	Revision flowchart	19
Figure 5	The ceiling showing 2 correct sliders, one low slider, and one high slider	...22

Defence Statement

I: Context

The role of the spectator(s)

There is a joke I remember hearing once, though the exact origin is lost to time and memory. A comedian goes onto a stage in front of hundreds of audience members, and recounts that the night before, he attended a rock concert. During the concert, the lead singer sang the first verse of the song, then held the mic up to the audience, who sang the chorus. The comedian continues, lamenting that the same trick doesn't work in comedy. To prove his point, he begins a joke: "A man walks into a bar", and holds the microphone towards the audience. The audience responds with laughter.

The humour, of course, is that "viewing is the opposite of knowing". (Ranciere 2009, 2) The audience — the passive spectators, gazers, observers — can't possibly play an active part in this performance. Any individual in the audience could very well have had a variety of continuations and punchlines to the joke's first line, but that is not the part they are playing. The part of the audience is a passive one. The performer acts, the audience spectates. In the same way that the audience will not respond to a joke setup with a punchline, Richard Schechner notes that in a performance of *Othello*: "No matter how involved or displeased, no one jumps onto the stage to stop the murder" (Schechner 2013, 104-5).

Ranciere refers to this relationship as the "Paradox of the Spectator". He claims that there is no theatre without a spectator, yet *being* a spectator is "a bad thing" (Ranciere 2009, 2). He continues to state that "to be a spectator is to be separated from both the capacity to know and the power to act" (2-3). Ranciere calls for "a theatre without spectators, where those in attendance learn from as opposed to being seduced by images, where they become active participants as opposed to passive voyeurs" (4).

There is no plot in *Othello* where Desdemona is not murdered, and it would seem that there *cannot* exist the show *Othello* where Desdemona is not murdered. If the audience has the capability to make a fundamental change to a performance, that performance would be by definition fundamentally changed.

There are performance practices which do exist as a fluid multitude of possibilities, expressed through interaction and improvisation. Janet Murray gives examples of improvisational groups, participatory dinner theater, and murder mystery

parties (Murray 1997, 43). Murray also includes role-playing games such as *Dungeons and Dragons*, as "theatrical in a nontraditional but thrilling way. Players are both actors and audience for one another" (42). All of these participatory performances, though, involve a *reduction* of some kind. Generally this reduction is in the cohesion, polish, and spectacle of performance. Very rarely in a D&D campaign are there live pyrotechnics, for instance.

There is another ubiquitous assumption in performance: that the audience is a *plural* body. Schechner notes that while the audience experiences a performance on an individual level, with the play "[resonating] in each spectator's life" (Schechner 2013, 104), that they also act as a collective, and "share subtle but unmistakable cues" (104). Unless otherwise noted, the word *audience* almost always seems to assume a plurality. While this is not entirely a necessity, experiments with single or limited audiences again involve a *reduction*. Theater Replacement's *Bioboxes*, from 2007, experimented with microperformances of a single performer and audience, where the scale, spectacle, and duration were all reduced to allow for the performative method to exist.

In 1987, a fictional technology was created which could challenge these assumptions without the discussed reductions. On *Star Trek: The Next Generation,* and later on other *Star Trek* shows, characters in the fiction have access to a device called the "holodeck". Murray describes the holodeck as "universal fantasy machine… a vision of a computer as a kind of storytelling genie in the lamp" (Murray 1997, 14), which provides "customized entertainment for a variety of tastes" (15). The holodeck creates an entire facsimile of reality, which can be interacted with by a single or multiple people in real-time. Holographic characters are essentially indistinguishable from actual humans.

The technology of the holodeck, that of a fully immersive interactive world, remains out of reach. However, by using the performative methods of role-playing games, combined with the speed of modern computing, we can see the emergence of a new performative method. This emerging field of video games allows us to have a singularly customized performance, where each individual audience member acts as both spectator and actor in their own performance.

Musically interactive systems

My background is in music, and it is through this lens which I approached this performative method. Research in generative music has, especially within the last decade as computing power has increased, toyed with systems that involve and integrate user feedback. Generally speaking, this integration takes one of two tacks: that of extreme abstraction, and that of no abstraction.

The systems with no abstraction are the easiest to conceptualize from a musical standpoint, as their appearance and interaction often resemble normal audio interfaces. Harris Wulfson's *LiveScore* (Douglas and Winter 2010) presents as a rotary MIDI controller. Each dial is labeled, via masking tape and pen, with a musical aspect such as "Duration" or "Range". As the user sets the dials, the parameters of the software are directly affected.

In the area of extreme abstraction there are works such as Jason Freeman's *Flock* (Freeman 2007). In *Flock,* a generative system tracks the location of the audience on a floor, as seen from an overhead camera. This data comes through numerous layers of abstraction. Dancers improvisationally choose movement patterns, which audience members interpret from visual cues, which they are free to interpret or ignore. This is then translated into a nonspecific graphical score, which musicians improvisationally interpret.

At performances of *Flock*, Freeman surveyed the audience, asking questions about their experience. The audience responded most negatively to questions about whether the performance "would have been different without them". In the comments, one audience member responded that they felt the performance was "very unorganized, chaotic, unstructured". Game designers and theorists Katie Salen and Eric Zimmerman describe an important concept in games of *rules*. They give several characteristics of rules: rules limit player action, they are explicit and unambiguous, they are shared by all players, they are fixed, binding, and repeatable (Salen and Zimmerman 2004, 134). They note that, somewhat paradoxically, by *limiting player action*, rules "constitute the structural system that *allows choice making to occur*" (150). In both *Flock* and *LiveScore*, many of these components are lacking. Most notably, in neither are the user's actions limited.

The idea of limits allowing choice making to occur is not specific to games or interaction; it is a commonly used technique in musical composition as well. Stravinsky famously stated that "The more constraints one imposes, the more one frees one's self" (Stravinsky 1947, 65). Cage relied on the structure of the *I Ching* in *Music of Changes* to allow for compositional decisions. Taken to an extreme example, the idea of simple tonal structures, or of Schoenbergian tone-rows, can be viewed as constraints which allow for meaningful choice making to occur. Using rules in interactive art allows this choice making to retain its creative focusing, but with the *audience* making the choices, rather than the creator.

II: Previous work

DJ Tower

In the spring of 2016, I created a musically generative interactive work, titled *DJ Tower*. *DJ Tower* was built to be an installation, with the interface and presentation designed to fit within a mocked-up arcade cabinet. The gameplay was modeled after a traditional *tower defense* game. Atari's 1990 arcade game *Rampart* introduced many of the common elements of the modern tower defense genre. The player builds structures, which act autonomously to defend an objective from enemies. As the player defeats these enemies and successfully defends the objective, they receive resources which allow them to further develop their structures.

The tower defense genre in many ways mimics a traditional musical arc: the player begins with limited resources, and as they make decisions about how and where to assign those resources, their resources grow. As their resources grow, their structures can be built and upgraded. As they approach the end of the game, the forces overwhelm the towers and there is a falling off.

I was not satisfied with *DJ Tower* by the time that it was installed, for a variety of reasons. During the development of the game, I first built the framework for the game, where I made decisions early on, both about gameplay methods and about the capabilities of the music generation. While this did unify the gameplay and the music, it also restricted both. The result of this was an unsatisfying gameplay which corresponded with unsatisfying music.

In addition to the gameplay and music being restricted by the early decisions, they were also restricted due to the attempt to combine two separate entities rather than building a unified experience. The musical feedback on the player's actions was removed by so many steps that the gameplay and music seemed almost to be unrelated. The user would play a game, and music would incidentally happen.

Perhaps the largest problem with the overall work was that in order to ensure that the player had access to the full experience, there was no difference between skillful play and random play. The rules, in this case, were improperly applied. The choices did not restrict the player actions, and so meaningful choice making could not occur.

These problems can generally be attributed to a lack of exploration and iteration in the development. I took these lessons as I began work on *The Audience of the Singular*.

II. The Audience of the Singular

Overview

Salen and Zimmerman describe the game development concept "Iterative design" as "a cyclic process that alternates between prototyping, playtesting, evaluation, and refinement" (Salen and Zimmerman 2004, 31). This approach is similar to almost any creative approach. Any form of composition involves a cyclic alternation between creating, experiencing, evaluating, and refining. Iterative design not only describes the process by which *The Audience of the Singular* was *created*, but also an underlying methodology of how its fundamental systems work, and even extending to the experience of playing the game itself.

The game tasks the player to play the part of a DJ. The player stands behind a virtual mixing board, on a virtual stage, in front of virtual audience, in a virtual room. An arranger system creates a series of suggested mixes for the music, which are relayed to the player via series of dance-floor-like lights. The player must interpret and experiment to find the exact meanings of the lights. As the player quickly mixes the music towards the arranger's suggestions, the audience becomes increasingly excited. As the audience becomes increasingly excited, the arranger suggests greater variations in mixing.

Presentation and Aesthetics

The aesthetic design of the project was chosen first, that of presenting as a pastiche of gaming and culture from the infancy of the current era of widespread, consumer-era video game consoles. This time corresponds generally with the release of the Atari 2600 in 1977, booming with the Famicom/Nintendo Entertainment System in 1983/1985, and ending with the advent of real-time 3d graphics in 1994 with the release of the Sony PlayStation.

The visuals were created to be evocative of early vector graphic displays, which were popular in arcade machines around the end of the 1970s and the beginning of the 1980s. This visual style was chosen out of several aesthetics from video games which existed during the chosen time. This style was specifically chosen out of the options because it also references the aesthetic of electronica around the early 1980s, specifically that of neon, contrasted with darkness.

As the design progressed, the pastiche became more impressionistic, and the range of history which I pulled from expanded. Rather than attempting to faithfully recreate any particular look, sound, or feel, instead I aimed to take enough from the source material to give a stylistic impression of the material. The decreased level of faithfulness in the recreation came as I further embraced the options afforded to me by modern techniques.

Quarter-circle forwards

Create in general, correct in specifics

The music generation engine behind The Audience of the Singular is called "Quarter-Circle Forwards", or QCF, for short. At its most basic, QCF uses a weighted Markov model, created from corpus-based learning, and then mediated by hand-crafted heuristics. QCFs generation first analyzes and learns music from the corpus, which follows the general design aesthetic and involves music from the video games of the late 1980s and early 1990s. After the initial analysis, QCF generates music to roughly approximate the corpus using weighted stochastic methods generated from the analysis of the corpus. After the generation, QCF then evaluates and refines the initial generated music, with its evaluation driven from my own musical evaluation of its initial output.

Using a real-time generative system was based on both pragmatic and artistic reasonings. Julian Togelius notes that algorithmically generating content "can augment the creativity of individual human creators", allowing small teams or individuals to "to create content-rich games by freeing them from worrying about details and drudge work while **retaining overall directorship of the games.**" (Togelius 2016, 3) (Emphasis added). Subset Games' *FTL: Faster Than Light* uses procedural content to create an experience in which each playthrough has a unique narrative storyline, mediated through both the player's actions and the generative system. Games which use generative content in this and related ways show that proper use of generative content can create unique, and most importantly, *personal* experiences. Essentially, generative content reinforces the *choices* which the audience makes, giving them more impact on the performance.

The generation, left to its own devices, had certain patterns which were decidedly unmusical, but which only appeared in specific and limited circumstances. This was difficult to deal with during the generation, as the logic powering the generation painted with a very broad brush. Because of the weighting towards general trends, any low-probability rough edge would eventually be compensated for. Many of the unmusical decisions, then, were not indications of systemic flaws, but mere coincidences. I was hesitant to create any rules which would curtail the ability of QCF to pick non-standard decisions when in a *majority of cases* those decisions made the music more interesting.

The solution which I eventually enacted was the addition of a final revision pass, which corrected these specific problems. The revision pass walks itself through the music multiple times, applying strict and specific rules. An added benefit of adding this revision pass is that it allowed me to further focus the methodology of the generation. Because specific problems would be addressed if and only if they manifested, the generation could lean more heavily into more open-ended, trend-based means. Essentially, this leads the generative system to be constructive and creative - guiding towards good decisions but doing nothing to prevent strange or "creative" decisions, and the revision pass stepping in only when the strange decisions manifest as bad decisions.

This description lays out the overall framework of QCF. For a more complete breakdown of the internal function of QCF, see Appendix A.

Audio

The music synthesizer was built into a PureData patch, which was loosely based on sound hardware from the late 1980s and early 1990s. As the general aesthetic expanded and became more impressionistic, so did the faithfulness to the original hardware. The synthesizer has a total of 7 instrument channels, and simply takes Pitch, Volume and Onset data from QCF, with a script within Unity controlling the tempo, and "playing" through the lines itself. For a more detailed breakdown of the synthesizer patch, see Appendix B

Arranger

The arranger class is the unifying element between the gameplay and the music. The arranger both powers the actual arrangement of the music, as well as creating the objectives for the player. The arranger was built using similar architecture to QCF: it generates at first using general trends, and then mediates specific choices in a revision pass. The arranger first creates a set of internal goals, an "ultimate mix", which it will attempt to move towards. Each time the player mixes according to the current suggested mix, the arranger moves one or more lines, depending on how well the player has been meeting the mixes.

Similarly to QCFs music generation, this open-ended, trend-based generation could result in poor decisions, given specific circumstances. These decisions are addressed in a revision pass of each suggested mix. The revision decisions, as in the generation, are based on my evaluations of the arranger through the iterative design process.

Gameplay

The moment-to-moment gameplay is based off of Mordecai Meirowitz' 1970 code-breaking game *Mastermind*. Eric Weisstein of *MathWorld* describes *Mastermind*: "In the game, one player is the codemaker and the other is the codebreaker. The codemaker secretly selects a code consisting of an ordered sequence of four colors, each chosen from a set of six possible colors, with repetitions allowed. The codebreaker then tries to guess the code by repeatedly proposing a sequence of colors. After each guess,

the codemaker tells the codebreaker two numbers: the number of correct colors in the correct positions and the number of colors that are part of the code but not in the correct positions".

In AOTS, the Arranger takes on the role of the codemaker. The mix is a series of 4 levels, each level can be one of 4-16 possibilities, depending on the player's progress.

Figure 1: Gameplay Screenshot

As the player mixes the music, the game responds in real-time with 3 numbers: The number of correct levels, the number of levels which are too high, and the number of levels which are too low.

These numbers are relayed via the ceiling, which is animated and arranged to resemble a typical dance floor. The ceiling contains 32 individually controlled lighted squares. These squares have a variety of arrangements which generally consist of a repeating pattern of 16 squares, divided into groups of 4 squares for each line. Each time the suggested mix is met, the ceiling pseudo-randomly assigns each group of 4 squares to directly correspond with one of the 4 sliders. If the level of the slider is below the suggested mix, the corresponding grouping of squares becomes blue, with the brightness increasing as the slider approaches the correct level. If the level of the slider is above the suggested mix, the corresponding grouping of squares becomes yellow, also increasing in

brightness with proximity to the correct level. If the level of the slider is within the suggested mix, the grouping of squares becomes green and bright.

If the player successfully and quickly matches the suggested mix, the audience becomes increasingly excited. As the audience's excitement rises, the game begins to increase the difficulty. It does this by widening the possibilities for the arrangers, increasing the range of the sliders, increasing the number of levels which the arranger changes each generation, etc. For examples and a more in-depth guide to the gameplay, see Appendix C.

Because the music and the specifics of the arrangement are both generated in real-time, rather than pre-determined, each playthrough of the game is a unique entity, bound by general similarities. Each audience member is not only singular as an audience member, but participates in the singular of the performance itself. A video demo of gameplay is available at http://at.sfu.ca/lFJuVX.

III. Reflection

The use of interaction necessarily involves giving up a degree of control. While I still feel that I have a significant artistic voice in both the final product and the generative system, this lack of control was quite noticeable. Because much of the content of *The Audience of the Singular* is gated behind skillful play, there is a risk that the player may not encounter the work as it is designed. Much of the play is designed around a few assumptions of the interaction, but in order to keep the audience as an active participant, these assumptions could not be forced. Finding this intersection of control and freedom for the audience remains a difficult target. To bring back the metaphorical example from Schechner, while I wanted to allow the player to stop Othello from murdering Desdamona, if the audience were to 'go rogue' and simply kill all of the characters at the beginning of the show, that would break the performance.

If there was one striking lesson which I took from the creation of this project, it was the similarity between crafting QCFs generative system, along with the arranger system, and simply writing music. The difference it seems is primarily that of, keeping in line with the project, the difference between the singular/plural, or the specific/general. There was still the loop for me of create - listen - evaluate - refine - repeat, though instead

of refining a specific choice, instead I would refine the cause of that choice. I often describe my work to those unfamiliar with the field as “teaching computers to write music”, and it often felt as though that was a perfect description.

Contrastingly, if there was one aspect of more traditional composition that I find myself missing in this new method of creation, it is that specific control which was necessary to give up. In many ways, my interest in generative music seems, to me at least, to be a natural extension of my nature as a highly technical composer. I enjoy creating exact and specific music, which is not so far from enjoying algorithms, which obviously need to be exact and specific. This level of control over the music is that which I missed the most in crafting the system. I could curb specific impulses, and encourage certain choices, but ultimately the result which reaches the audience, especially with the interaction, is far less of a controllable outcome.

Murray discusses the idea of “agency” in games, defined as “the satisfying power to take meaningful action and see the results of our decisions and choices” (Murray 1997, 126). Creating this satisfying power is a difficult prospect, though a necessary one to create an artistic experience which fulfills the aim of bring the audience into an authorship role. A helpful conceit of *The Audience of the Singular* for creating this feeling of agency was that while a DJ in the real world can technically mix however they want, they too are generally attempting to mix towards their audiences tastes. Because of this audience-focused real-world equivalence, I found both in my experiences and in feedback I received during testing, that players did feel as though they were taking meaningful action during gameplay, even though the specifics were actually controlled by the arranger.

I distributed the game on itch.io, which is a distribution front for independently developed games. While it was in development, I also consulted other game developers, and received informal feedback from a general audience of game players online. While their feedback was appreciated, it mostly served to reinforce my own thoughts and feelings about the project. The most useful part of the feedback which I received was mainly to confirm that my personal evaluation of the project was not overly divergent from my intended audience’s reaction. Ultimately, the opinion and feedback that I relied

on the most was my own. During development, I was in a constant loop of playing the game and evaluating each component while making changes.

In as far as *The Audience of the Singular* seeks to bring its singular audience into an active role in a unique and personal experience, I feel that it was successful. There certainly are aspects which remain weak, with fixes ranging from simply having more time and resources to completely overhauling major systems. In terms of satisfying the game's raison d'etre, though, the core concept and exploration, I feel that it was ultimately successful.

Looking forwards, there is room for further exploration of these concepts. I hope in the future to continue to explore the uses of generative art, both musical and other game content. I hope to continue to explore generative art specifically as a method for bringing the audience into an active role in the emerging performative field of gaming.

Works Cited:

Barret, Douglas and Winter, Michael. 2010 “LiveScore: Real-Time notation in the music of Harris Wulfson”, Contemporary Music Review. Vol 29, Iss. 1.

Bigo, L. and Conklin, D. (2015). A viewpoint approach to symbolic music transformation. In 11th International Symposium on Computer Music Multidisciplinary Research (CMMR 2015), pages 56–70, Plymouth.

Conklin, D., and Witten, I. H. (1995). Multiple viewpoint systems for music prediction. Journal of New Music Research, 24, 51–73.

Eigenfeldt, Arne. 2009. “The Evolution of Evolutionary Sofware: Intelligent Rhythm Generation in Kinetic Engine.” Proceedings of EvoMusArt 09, the European Conference on Evolutionary Computing, Tübingen, Germany.

Eigenfeldt, A. and Pasquier, P. (2013). Considering vertical and horizontal context in corpus-based generative electronic dance music. In Proceedings of the 4th International Conference on Computational Creativity (ICCC 2013), pages 72–78, Sydney, Australia.

Freeman, Jason and Godfrey, Mark. 2008 “Technology, Real-time Notation, and Audience Participation in Flock,” Proceedings of the ICMC, Belfast, UK.

Ma, Justin and Davis, Matthew (2012). FTL: Faster Than Light. Subset Games.

Murray, Janet H. 1997. Hamlet on the Holodeck: The future of narrative in cyberspace. Cambridge, MA: The MIT Press

Puckette, Miller 2016 PureData version 0.47.1, San Diego, CA. Available from http://www.puredata.info

Ranciere, Jaques. 2009. The Emancipated Spectator. London, UK: Verso.

Roddenberry, G., Hurley, M., Baron, M., Barry, P., Beimler, H., Black, J. D. F., Davis, D. D., Star Trek, The Next Generation... Paramount Pictures Corporation,. (2007).

Salen, K and Zimmerman, E. 2004. Rules of Play – Game Design Fundamentals. Cambridge, MA: The MIT Press

Schechner, Richard. 2013. Performance Studies: An Introduction. 3rd Edition. New York, NY: Routlege

Shaker, Togelius, and Nelson (2016). Procedural Content Generation in Games: A Textbook and an Overview of Current Research. Springer. ISBN 978-3-319-42714-0.

Stravinsky, Igor. 1947. “Poetics of Music in the form of six lessons” Cambridge, MA: Harvard University Press

Theatre Replacement 2007, January 3. Bioboxes. Live performance in High Performance Rodeo, Calgary AB.

Weisstein, Eric W. "Mastermind." From MathWorld -A Wolfram Web Resource. Retreived from http://mathworld.wolfram.com/Mastermind.html

Appendix A: Cyborg Collaborators

"On May 25, 2006, 70 Russian pilots sat tired and bleary-eyed in their battleships near the first moon of the fourth planet in a star system so remote it was known only by the obscure designation 'C-J6MT'. They had dug in their heels, huddled around their last stronghold…and were preparing for the fight of their lives.

…ships warped in on the position of the 70 defenders…The siege of C-J6MT had begun"

– From *Empires of Eve: A history of the great wars of Eve Online* (Groen 2016)

Introduction

"Spectators experience a theatre piece" (Schechner 2013, 104-5). In a simple explanation of a simple action, Schechner reveals a pair of assumptions which exist in performance. One of these assumptions, that of the unidirectional nature of theatre and performance, is the topic of the same section of the book. Schechner notes that, starting in the 1960s, the "rules governing audience participation were often vague or deliberately ambivalent", and that there has been a growth of audience interaction. The other, less obvious assumption, however, remains unchallenged. Even in the discussions concerning the audience, the *plurality* of the audience/spectators remains unchallenged.

The plurality of the audience seems to be required of performance on a level of mere practicality and economics. The cost of even a small troupe of actors, let alone the technical support and venue, would make a singularly-spectated performance untenable. The unidirectional audience/performer relationship still remains true in many forms of performance, as well as many other forms of art. Schechner mentions, in the discussion of a performance of *Othello*, that "No matter how involved or displeased, no one jumps onto the stage to stop the murder". The audience, no matter how involved, generally still

does not drive the performance. This also seems to be a practical matter: The performers most likely have not prepared for every one of the seemingly infinite ways which an audience *could* drive a performance, if they chose to do so. Because *Othello*, like most works of theatre, has a pre-determined plot, the audience can only passively "experience".

With the advent of computationally generative art, there exists the possibility for reactive algorithms to break this deterministic rigidity, and allow for an audience of the singular to experience art of their own creation. In 1987, the aspirational form of this idea was given voice in the form of the Holodeck, on the series *Star Trek: The Next Generation*. Janet Murray notes that "[Experiences in the Holodeck] provide *customized entertainment* for a variety of tastes" (Murray 1997, 15) (emphasis added). The technology of the physical synthesis of the holodeck remains yet out of reach. The customized entertainment, though, does not.

The burgeoning field of Video Games have the potential to realize this relationship, not of spectators and performer, but of the single entity of audience/performer. Using techniques of algorithmic generation, games can provide a space for players to take an authorship role over the performance, acting as creator, performer, and spectator.

My research explores these concepts, through the use of musically generative algorithms, which the audience/performer interacts with via a video game. The audience/performer collaborates with an AI system. They create, perform, and experience their own unique music, each time they play.

Abstraction and Procedural Generation

There are two interacting components of games which enable the emergence of the audience/performer. These components are *Abstraction* and *Procedural Generation*.

Abstraction is a term borrowed from Computing Science. John Guttag writes that "The essence of abstractions is preserving information that is relevant in a given context, and forgetting information that is irrelevant in that context" (Guttag 2016, 109). Abstraction is a tool which presents the programmer or user with the ability to manipulate and interact with information that they do not need to fully see or understand. For instance, consider the following C# code snippet:

```
class Circle
{
    public float Area(float Radius)
    {
        return (float)(Math.PI * Math.Pow(Radius, 2));
    }
}
```

This code creates an object class of "Circle", and includes a function which can return the Area of the circle based on the Radius. Elsewhere in the same program, this code could be accessed by the following function:

```
void CreateCircle()
{
   var Circle1 = new Circle();
   var Area = Circle1.Area(4.0f);
}
```

In this function, the programmer *does not need to know* how to find the area of a circle. When a "Circle" in the context of the program is created, it knows how to find its own area. Games can essentially be thought of as this concept taken to an extreme. When a player inputs a command into the game via the User Interface, a series of functions and

algorithms are called, which then present back an abstracted response, also via the UI. The player of a game does not need to understand any of the underlying computations occurring; the game surfaces both the inputs and responses in an easy to understand format.

The second major component which enables the customizability of the performance is Procedural Generation. Julian Togelius defines Procedural Generation in games as "creating game content automatically, through algorithmic means" (Togelius 2011, 1). Because procedural generation is created through algorithms, rather than manual production, massive amounts of content can be created in seconds or less, with essentially an infinity of possibilities. Togelius continues his discussion of the strength of procedural generation: "Imagine a game that never ends — wherever you go, whatever you do, there is always something new to explore, and this new content is consistently novel while at the same time tuned to your playing style" (2).

With these two factors, the methods by which games can provide the customized experience takes form. By simply interacting with an abstracted interface, a player of a game can manipulate and modify the sophisticated procedurally generative algorithms which the computer is running.

Interaction with Generative Musical Systems

In the last 15 years, there have been many generative musical systems which use audience interaction in some way. There have been a wide variety of ways in which audiences can abstractedly interact with the systems. Sometimes, this interaction is hidden from the audience, such as in Jason Freeman and Mark Godfrey's *Flock* (Freeman and Godfrey 2008). In *Flock*, the audience wears custom hats, which have LED lights mounted on them. The audience can alter the music generation by moving around the room, and by turning their personal light on or off. Both of these actions can be done at will. The generation system then takes the location data of the lights and turns it into a graphical score, interpreted by live musicians.

Some systems present the user with all of the information about their interaction, as seen in *LiveScore* (Douglas and Winter 2010), this interaction simply presents the audience with several labeled dials, which directly change parameters of the generative algorithm.

LiveScore

Both of these examples utilize audience interaction. However, their ability to create performances with *meaningful* interaction is limited at best. Freeman notes that when asked about their experience with *Flock*, the lowest audience responses involved asking if the audience felt that the performance would have been different without them, followed by whether they felt as though they were creative, and if they understood how they helped shape the performance (4).

Systems with similar interactions are common in generative music, and often have similar reactions from audiences. The field of Game Design explains this with the concept of "Rules". Katie Salen and Eric Zimmerman state that "All games have rules, and rules are one of the defining qualities of games" (Salen and Zimmerman 2004, 134) and define rules as having the following aspects:

- Rules limit player action
- Rules are explicit and unambiguous
- Rules are shared by all players
- Rules are fixed
- Rules are binding
- Rules are repeatable

Without rules, these systems become *toys*. They are interesting experimentally, though they do not have a meaningful output. Salen and Zimmerman continue to discuss why limiting player action is so vital to meaningful interaction: "Rules constitute the structural system that allows choice-making to occur" (150). By borrowing the use of rules to structure meaning in the interaction, a wider array of actual choices can be made by the audience/performer of an interactive generative system.

Independent Game Development

On March 2nd, 2015, Epic Games changed the subscription model for their "Unreal" game engine. Unreal is a professional-grade game engine, used in many large-scale, major published game releases. Epic Games changed the model to be completely free, with their income coming from a small royalty over a certain profit. On the next day, Unity3d released their "Unity 5" engine, the 5th iteration of their game engine. Unity before had been used primarily for independent developers, students, and tinkerers. Its subscription model was free, though only with an incredibly basic version of the engine. Unity 5 changed the subscription model, increasing the basic, free version of Unity 5 to be arguably superior to the "Professional" version of the previous iteration. Unity 5 can be used to develop and release professional-level games, with additional licenses only required if a certain profit is made.

These changes in the landscape game engines made it incredibly easy for anyone, with or without previous knowledge of coding or game development, to easily begin creating games. Additionally, distribution networks have made it increasingly easier to *release* these games. In 2013 the website itch.io launched, which allows developers to post a game to the internet for free, set their own price or price range, even set the revenue split between developer and itch.io. As of writing this paper, itch.io has 49,219 independently developed games in their catalogue. The end result of these changes has been that creating and publishing games is now possible from any background, and by individuals and small teams. While originally games were made by single developers (Robinett 1979), development team size has rose steadily over time. 2015's *Assassins*

Creed Syndicate, for instance, had a development team estimated at over 500 people (Ubisoft 2015).

QCF(Quarter-Circle-Forwards)

My work over the last several months has been on the development of "QCF", or "Quarter-circle-forwards", a Machine-learning system for generating new music. QCF is built in C#, to allow it to be seamlessly integrated into a game built using Unity 5. At its most basic, QCF uses a Hidden Markov Model combined with Heuristic filtering to create new music.

The first step in QCFs algorithm is the import and parsing of the corpus via the "AnalysisProgram". The corpus consists of music from video games of the 1980s-1990s, such as *Street Fighter II* (Capcom 1991), and *Castlevania* (Konami 1986). These files are stored in .csv (character-separated variable) files.

As an example of QCF in action, take the following simple musical line:

This line would translate to a .csv as the following table:

2	10	Note_on_c	0	60	81
2	986	Note_off_c	0	60	0
2	1054	Note_on_c	0	62	77
2	2001	Note_off_c	0	62	0
2	2069	Note_on_c	0	64	79
2	3016	Note_off_c	0	64	0
2	3084	Note_on_c	0	65	77
2	4028	Note_off_c	0	65	0
2	4100	Note_on_c	0	67	76
2	8124	Note_off_c	0	67	0

QCF first parses this data into objects as understood by C# and simplifies to the important data. The data QCF takes out are in columns 1, 2, 3, and 5. This data involves:

1.Which line the trigger is part of.

2.The time in Tick, where a Tick is approximately 1/256 of a sixteenth note

3.Whether the trigger is on or off

5.The Pitch, in midi value, of the trigger.

After parsing the data, QCF creates a complex data-storage objects `Note` and `VerticalSlice`. Each contains similar data, but `Note` is for each note, and `VerticalSlice` for measures.

Note Object

```
public class Note
{
    //Builder-set variables
    public int Pitch; //MIDI pitch of note
    public int Duration = 0; //Time until the note stops
    public int TimeToNext = 0; //Time until the next note starts
    public int Contour = 0;  //Whether the pitch goes up(1) or down(-1) from last
    public int totalContour; //Total addition of contour values
    public int PlaceInMeasure; //Place in the measure
    public int totalTime; //Time in 16ths from beginning
    public List<int> LastPitches = new List<int>();  //Pitch of Last 5 notes
    public List<int> LastDurations = new List<int>();  //Duration of last 5 notes
    public List<int> LastTimeToNext = new List<int>(); //TimeToNext of last 5 notes
    public List<int> LastDeltas = new List<int>(); //Last 5 distances between notes

    //Immediately Derived Variables
    public int ContourOfLastNotes; //Total addition of contour values
    public int DistanceFromLast; //Distance from last note
    public int BroadContourSoFar; //Sum of DistanceFromLast which exists so far.
    public int DurationDelta; //Difference between Duration and TimeToNext

    //Derived variables for vertical slices
    public int DistanceFromBass;  //Distance from current pitch of bass note
    public int DistanceFromRoot; //Distance from average pitch of bass line
    public int DistanceFromMelody = 0;

    public void SimpleNote(int pitch)
    {
          Pitch = pitch;
    }
 }
```

VerticalSlice

```
//Class for Vertical Slices for analysis
public class VerticalSlice
{
    public int ID; //ID of slice, for bookkeeping

    public int SliceLocation; //Measure
    public int PitchAtStart; //Absolute contour at START of slice.
    public List<int> PitchesInSlice; //Contour ACROSS slice.
    public int DistanceFromBass; //Distance from Bass at START of slice.
    public List<int> AvgDistanceFromBass; //DistanceFromBass ACROSS slice.

    public int DistanceFromMelody; //Distance from Melody at START of slice.
    public List<int> SliceDistFromMelody;//Distances from melody ACROSS slice.

    public List<int> AcceptableDistancesFromRoot;
    public List<int> AcceptableRoots;

    public int SliceMax, SliceMin;
    public int BassDistMax, BassDistMin;
    public int MelDistMax, MelDistMin;
}
```

These data objects are where QCF generates its own version of Musical Theory; where it constructs "rules" of how music is constructed.

QCF also creates the core of generation logic: Markov Chains. For pitch, the Markov Chains are simple, they are collections of `MarkovCell` objects, which contain a list of integers as a "Seed", and a List of pairs of integers, representing the potential result, and the number of occurrences in the corpus for the results. The final `MarkovCell` created from the pitches of the example provided above would have a Seed of {60, 62, 64, 65} with a Result of {(67, 1)}. The Markov Chain builds to a seed length of <=5 pitches. For rhythms, similar logic is used, based on a modification of Arne Eigenfeldt's Kinetic Engine Rhythmic generation (Eigenfeldt 2009, 5). Rhythm is constrained to a "beat", which is stored as a pointer to a List of all possible subdivisions of a beat. After that, it creates a Markov Chain similar to the melody chain, with a seed length of <=3. The final Markov chain for rhythms from the example would be a single Cell with Seed of 3 Quarter notes, and result of a Quarter note, with weight 5, and a tie coefficient of 0.2. The Whole note would enter the program as 4 Quarter notes, with the first 3 tied to the next.

After the Analysis program has completed, the Generative program begins. The generative program basically reverses these steps, using the Markov Chains and pseudo-randomness to stochastically generate new melodies. The new Melodies are stored in a simplified variation of the `Note` class. While generating from the Markov chain, the generative program also checks the `VerticalSlice` objects for each measure as well, and changes the weights of the potential results, to apply QCF's interpretation of Theory. The generative program generates the Bass line first, where it uses a modification of the

Markov Chain to create chord progressions and phrases of between 2-8 measures. After that, it creates two pairs of lines: 2 Melody lines and 2 Rhythmic lines. At this phase QCF also creates 3 alternate pieces, each of a differing rhythmic density, filtering notes so that the melody is retained, but rhythmically simplified.

Upon finishing the generation, a Heuristic program goes through the melodies and applies heuristics based upon my own knowledge, from my background as a musician. These heuristics include:

1. Forcing nonharmonic tones into the mode, if they do not meet standard musical nonharmonic tone archetypes (Appoggiatura, Escape Tone, or Neighbour) or if they occur on beats 1 or 3 of a measure.
2. Creating micro-phrases of 2-8 beats that repeat, replacing the next notes
3. Shortening some longer notes, and leaving a rest, to give a pseudo-phrase-break occasionally
4. Enforcing that the secondary lines should be always below the primary lines

QCF Development

QCF's development to now has gone through 11 "Generations", where each generation is defined by a major change in a system. These Generational shifts can be structural or musical, and so the shifts are not always noticeable in the generated output. For brevity's sake, I will mainly focus on a meta-generational window, points where the change in the output is noticeable and obvious. Throughout all of these generations there are some aspects which continued: I was constantly revising the code to perform better and be more organized, I was constantly and continue to add to the corpus, and the parameters are always being tuned.

The first and second generations of QCF were based on a theory of using a delta-focused approach rather than absolute pitch. Rather than understanding an intervallic jump as two pitches, it would instead understand the jump as the difference between the two pitches. The biggest problem with this approach was that in order to make modal music, pitches would, at some point, need to be hard-coded into an understanding of music theory. This understanding would probably be derived implicitly from the pitches in the corpus, which would mean that eventually the system would be considering absolute pitch anyways, but would need to do far more calculations to reach the end point. This was scrapped due to both not doing what it was originally intended to, and also due to creating music which sounded very bad.

By the third generation, QCF began generating based on pitches, and also had a rudimentary understanding of music theory. It did not yet have an understanding of Rhythm. Generations 4 through 8 were mainly focused on creating the understanding of rhythms. Because QCF stores each note as a Pitch, Duration, and Time until the next

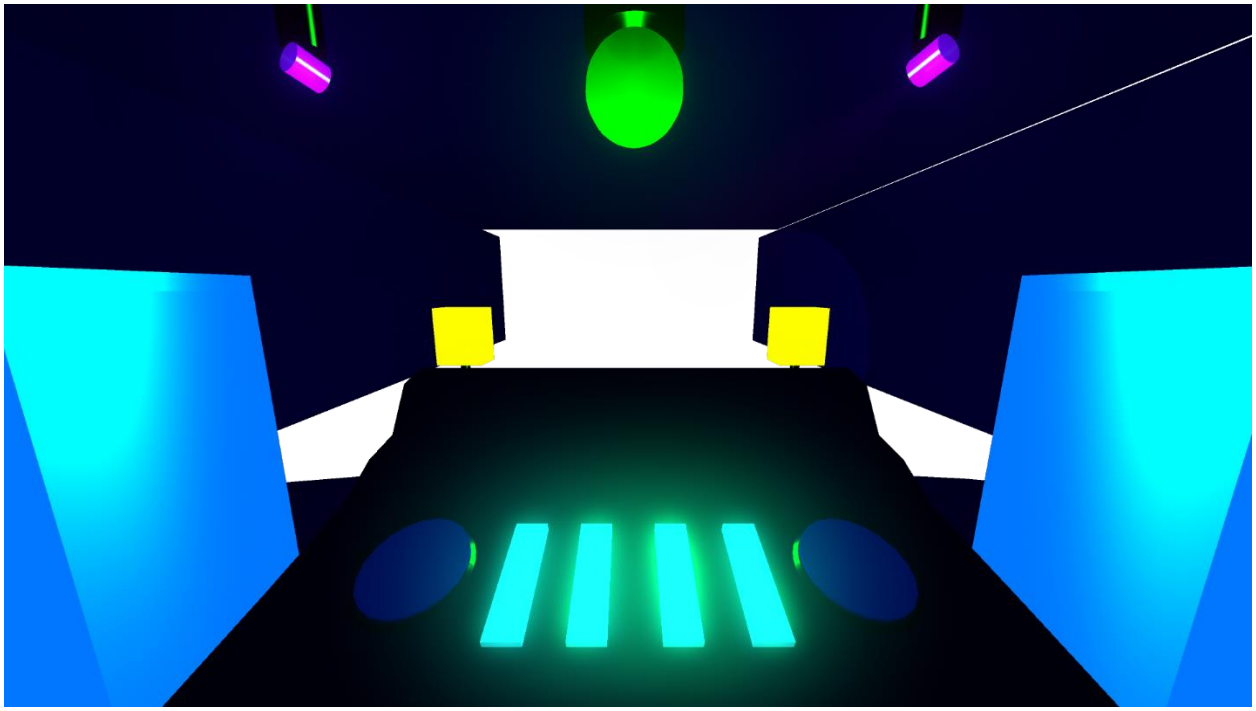
note, a difference in duration and Time to Next allow it to express rests, and also more complex rhythms such as syncopations. These generations also created the density filtering algorithm. The 9th generation also implemented a phrase "reset", which clears the Markov generation every 2-6 Measures.

At the end of the 9th generation, QCF had an understanding of music. It could analyze and produce music based on the parameters which it watches, and was correctly creating lines. The music often sounded chaotic and unfocused though. Secondary lines would often cross the primary lines, which often ended up removing the feeling of contour. Additionally, because QCF did not understand concepts such as harmonic/nonharmonic tones, nor did it understand strong/weak beats, it would often have pitch choices which were very grating.

My attention then mainly turned to increasing the quality of the music, rather than simply getting the mechanics complete of the basic learning system. Generation 10 and 11 implement the Heuristics program, which was not part of the original design. The most recent version is the 11th generation, which now is the full generation system. Recordings of QCFs output are available at https://soundcloud.com/cale-plut.

The 12th generation, which is in progress, is an overhaul of the construction of the system, separating the Analysis program into a more separate entity, to set up for the creation of the interaction: The game currently titled *DJ Simulator*.

DJ Simulator and the future



With QCF being almost complete in terms of its major structure, I'm beginning to turn my attention to the next part of this project: The interaction with the system, allowing the realization of the ideas which I've laid out. *DJ Simulator* will use QCFs generative system to create a unique piece of music, which the player will "mix", while attempting to entertain their simulated audience.

The player will have a limited amount of "power", a resource which they will use to increase the density, volume, and effects of each of the 4 major line archetypes: Melody, Rhythm, Bass, and Drums. The game will algorithmically generate audience groups with varying "tastes", which the player will attempt to match by applying the "power" to the correct lines. As the player mixes the music, the environment will also change, increasing the intensity of the lights and animations.

The full game will unfold over a series of waves, and in each wave the player will be given more power, allowing them to mix more. Similarly, the audience will have greater demands. Each wave will be about one minute in length, allowing a simple and short piece of music each time. The player will have the option to "freeze" track generation at will, and at any time. If they do not freeze the generation, then at the beginning of each wave, each track will be re-generated. If the player does freeze a track, then that track will be repeated during future waves. Each track will consist of numerous phrases, which can be arranged into a form which will be generated each repeat. Because the player is restricted only to general "power" levels and a yes/no binary on each line, they have the tools to meaningfully interact with the music on *a* level. They can consciously make *some* choices, while the system enforces the choices which are important for QCF to maintain control of.

By using these tools, the player will be able to build up their full piece of music, picking and choosing their favourite parts, and working with the system to create their "final" piece. When the player has finished the game, in either a loss or win state, the music will stop, and be un-savable by the user. Each performance is unique and ephemeral, existing only as long as the player can interact with it.

To paraphrase Schechner, I hope that when *DJ Simulator* is played, three actions occur simultaneously:

- A computational algorithm creates new music
- A person plays a game
- A unique and customized experience is performed.

Specifically, I hope that these three things happen in a way in which only tapping into the power of video games can allow for. *DJ Simulator* is, on its face, a few different things. It is simply a game in which a player allocates limited resources to achieve a goal. It is an interesting design presentation for interaction with some elements of a generative system. Hopefully, and most importantly, it is a vessel for an infinity of unique performances. It aims to be a mediator for a cyborg-collaborative multitude of personalized musical concerts. I hope through this work to, rather than "Spectators experience a", create a work where "Spectator/Performer creates/experiences infinite".

Works Cited

Barret, Douglas and Winter, Michael. 2010 "*LiveScore*: Real-Time notation in the music of Harris Wulfson", *Contemporary Music Review*. Vol 29, Iss. 1.

Capcom. 1991. *Street Fighter II: The World Warrior*. Designed by Akira Nishitani and Akira Yasuda. Chuo-Ku, Osaka : Capcom.

Eigenfeldt, Arne. 2009. "The Evolution of Evolutionary Sofware: Intelligent Rhythm Generation in Kinetic Engine." Proceedings of EvoMusArt 09, the European Conference on Evolutionary Computing, Tübingen, Germany.

Freeman, Jason and Godfrey, Mark. 2008 "Technology, Real-time Notation, and Audience Participation in Flock," Proceedings of the ICMC, Belfast, UK.

Groen, Andrew. 2016. *Empires of Eve: A history of the great wars of Eve Online*. Self-published.

Guttag, John. 2016. *Introduction to Computation and Programming using Python: With application to understand data.* Cambridge, MA: The MIT Press

Konami, 1986. *Castlevania.* Developed by Konami. Minato, Tokyo : Konami.

Murray, Janet H. 1997. *Hamlet on the Holodeck: The future of narrative in cyberspace*. Cambridge, MA: The MIT Press

Robinett, Warren. 1979. *Adventure*. Published for the Atari 2600. Information and playable version available at http://www.warrenrobinett.com/adventure/index.html.

Salen, Karen and Zimmerman, Eric. 2004 *Rules of Play – Game Design Fundamentals* Cambridge, MA: The MIT Press

Schechner, Richard. 2013. *Performance Studies: An Introduction. 3rd Edition.* New York, NY: Routlege

Togelius, Julian; Yannakakis, Georgios; Stanley, Kenneth; and Browne, Cameron. 2011. "Search-based Procedural Content Generation: A Taxonomy and Survey" *Computational Intelligence and AI in Games IEEE Transactions on, vol. 3*, pp. 172-186.

Ubisoft Quebec. 2015. *Assassin's Creed Syndicate*. Produced by Francois Pelland, written by Jeffrey Yohelm. Ville de Quebec, QC : Ubisoft.

Appendix B: Quarter Circle Forwards

QCF analyzes and creates in a 5-line arrangement, which I arrived at from my own early analysis of the corpus. The music of the corpus almost always had 3 primary pitch-based lines: a bass line, a harmonic/rhythmic support line, and a melody line. In more complex pieces, the melody and support lines would contain dyads with the same rhythm. The 5-line arrangement consists then of 2 Melody lines, 2 Support lines, and 1 Bass line. For the doubled lines, they have the same rhythms, and may or may not have the same pitches.

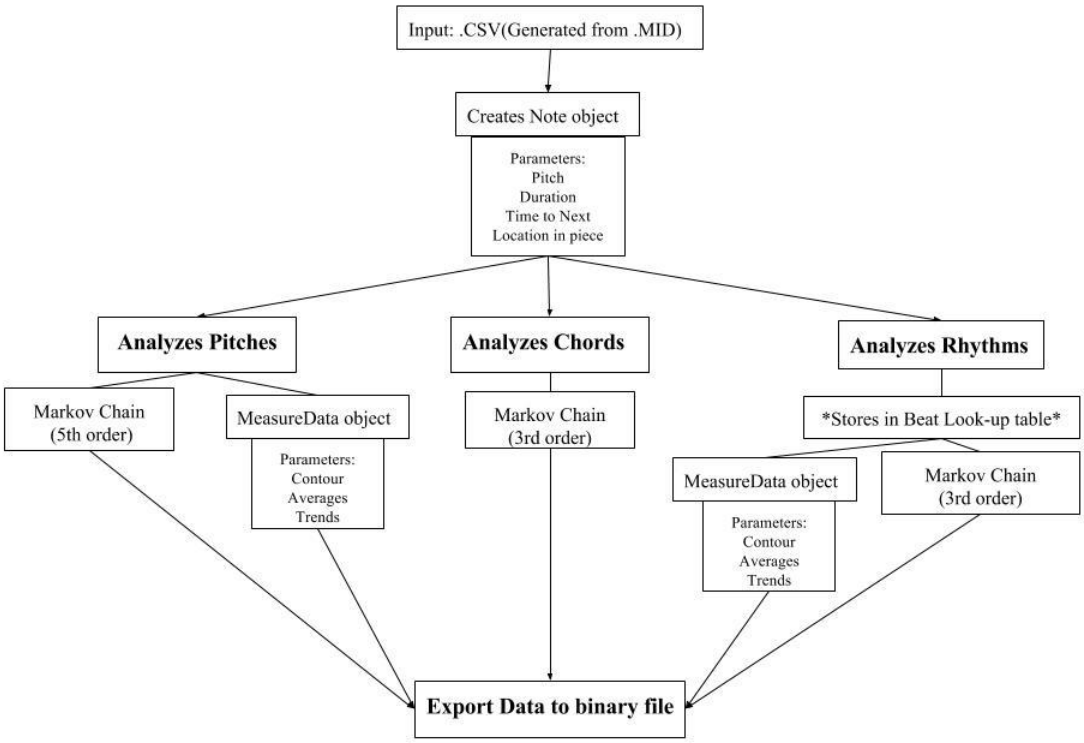
The design behind implementing the analysis program came from Darrell Conklin's "Multiple Viewpoint Systems for Music Prediction". Conklin describes a "collection of independent views of the musical surface each of which models a specific type of musical phenomena". While Conklin uses the viewpoints to *predict* music, the method can easily be used for generation. The Analysis Program creates a 5th-order Markov chain of pitches from the corpus, and then also creates complex "MeasureData*"* objects. The MeasureData class contains data on general trends of independent views, both across and within measures, across all pieces in the corpus. The Markov chain was chosen because the system needed to run very quickly, so an O(n) algorithm seemed best. Another reason for choosing Markov generation is that I have used it before, in *DJ Tower* and other projects of my own. Markov generation has also been used elsewhere in other corpus-based machine-learning systems, such as GESMI (Eigenfeldt and Pasquier, 2013). In QCF, if a cell with seed of length n is chosen and only has a single potential result, the seed of n-1 is also checked, and if there is no cell from even a first order chain with more than one result, a random cell is chosen and given a modifier of 50% on top of all of its other weights.

The rhythmic analysis follows a similar tack to the pitch analysis, though with an additional data point, and through a slightly obscured process to the raw pitch values. Storing rhythmic data for individual notes can often result in a loss of beat cohesion. To counter this, I used a modified version of Arne Eigenfeldt's Kinetic Engine's Rhythmic Generation. Rhythms are stored in a look-up-table of all possible rhythmic divisions of a single beat. QCF stores rhythms in a 3rd order Markov chain of integers which point to the look-up-table. The additional data point which is added to rhythm data is that of a

"tie", which allows for rhythms which are longer than a single beat, and also allows for syncopations. Rhythmic trends are also stored in the MeasureData object.

Chord progressions are handled in another slight alteration of the basic Markov model. During analysis, QCF determines a "chord" by finding the most populous bass note in a measure, and then finding each note which is in the other lines that are within Aeolian mode, adjusted for octave. These chord objects are then stored in their own 4th order Markov chain.

Fig 2: Analysis flowchart



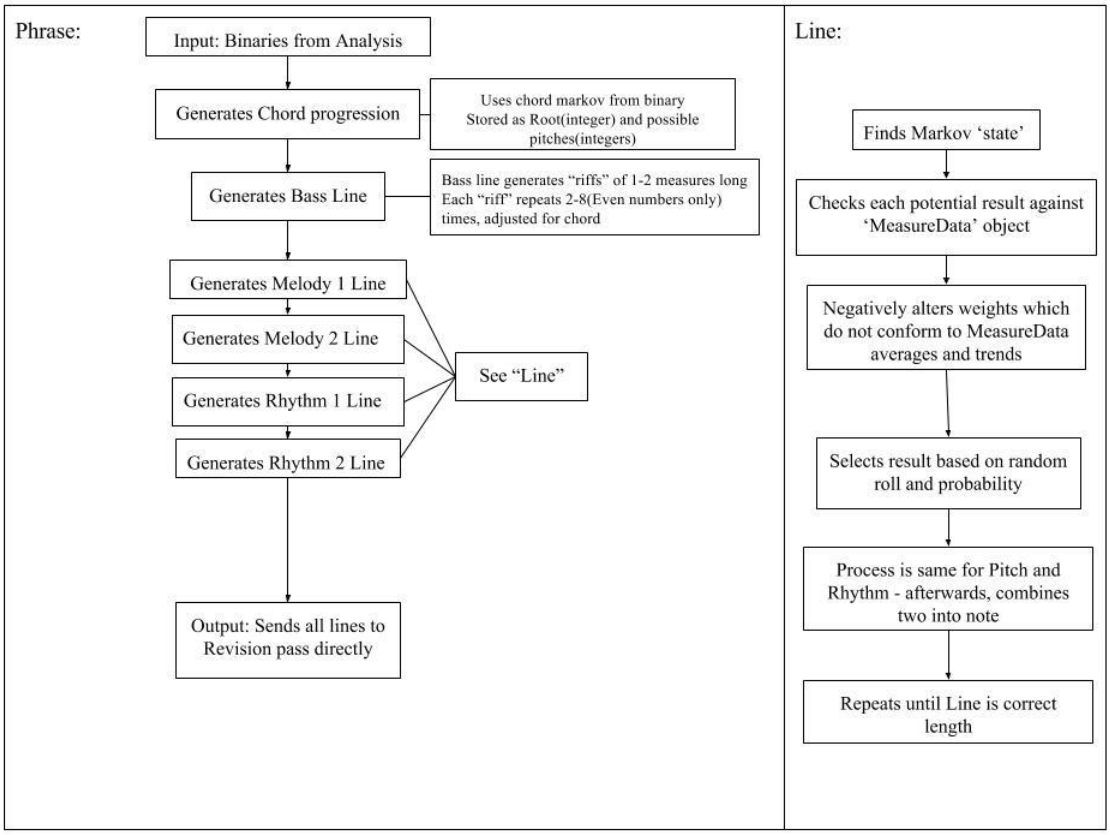
The generative program itself, at a very basic level, simply reverses the analysis program, randomly walking over the data. When choosing a note, the generation compares the previous notes to the Markov chain, and selects a "cell" from the chain. It then compares each potential result to the MeasureData's trend information and the chosen chord, and alters the chance of picking each result based on how closely that result fits into the general trends.

Because some lines depend on each other in their data, this is done in a specific order. QCF first simultaneously generates a bass line and a chord progression. The bass line is unique in that it will generate a single measure, and then repeat that measure, adjusted for the chord, between 2 and 8 times. This was done to create a cohesive sounding bass "riff" rather than creating simply a lower melody line. After the bass line is created, QCF creates a primary melodic line, and then a secondary "support" melodic line. After the melodic lines are created, it creates the primary and then secondary rhythmic lines. All lines depend on the chord, both primary lines depend on the bass line, and both secondary lines depend on their primary line.

This process is done a total of 4 times per generated phrase. During the course of the game, the player will be mixing the volume. In order to accentuate the difference in the volumes, each line has a full version, as well as 3 rhythmically filtered versions. Each version simply takes the pitches of the original version and places them over a newly generated rhythm that prefers longer tones and more ties. This process becomes more extreme for the least-dense versions. This filtering preserves the placement of notes *in time*, regardless of where they are in the melody. That is to say, if the line contains the pitches C-D-E over quarter notes, the filtered version may replace the C duration with a half note, and then entirely skip the D, playing the E next. This way, the overall contour and line is preserved, at the potential cost of the individual intervals.

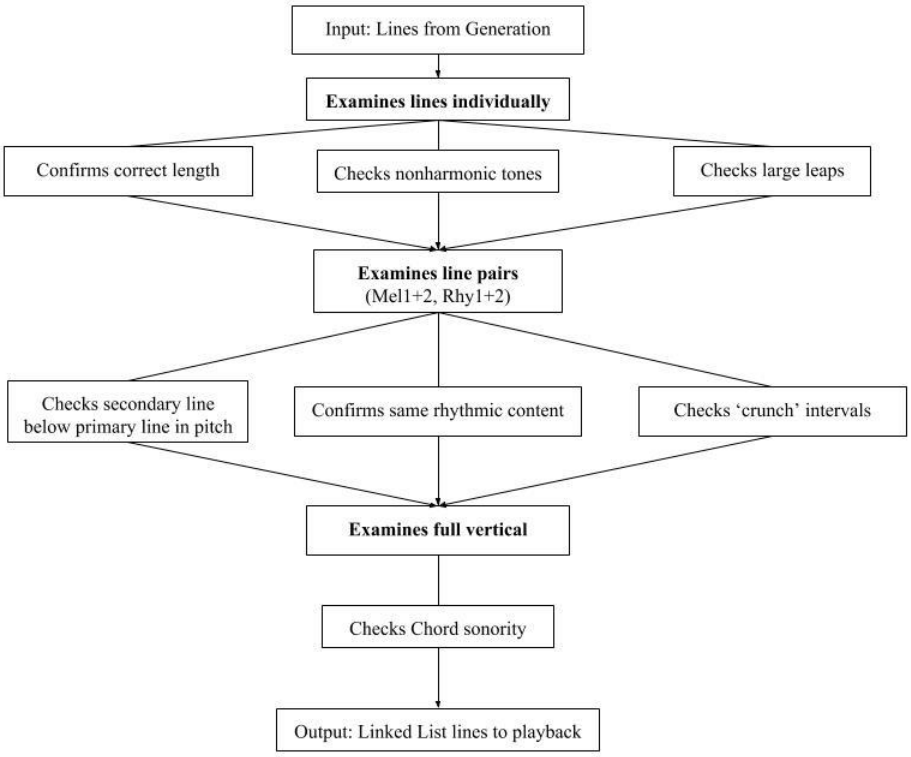
For the larger form, I took general inspiration from Conklin's "symbolic music transformation", where an existing template is used to structure the composition of a piece, rather than generating the form. The possible musical forms are stored in a list, and one is chosen randomly at the beginning of the generation. The generation then creates each phrase of a specific length, with the introductory phrase and a C phrase, if existent, half the length of the A or B phrases. Once the phrases are generated and the form is chosen, it is a simple matter of plugging the generated phrases into the correct form and walking through the sections until the end of the piece.

Figure 3: Generative flowchart



After the generation has completed, QCF then walks through the generated music and applies the revisions. These revisions were taken from my personal expertise as a musician, and act to 'smooth out' the rougher edges which the open-ended generation can result in.

Figure 4: Revision flowchart



Appendix C: PureData Patch

The PureData patch, which acts as the music synthesizer for the game, is based loosely on the sound hardware of the Famicom/Nintendo Entertainment system, the Rioch 2A03. In order to have greater flexibility in channels and sounds, I included Konami's sound expansion controller, the Virtual ROM Controller 6, or VRC6 chip. The original Rioch 2A03 chip was a PSG with 4 channels. For melody and harmony, it had 2 pulse waves, with duty cycles of 12.5%, 25%, 50%, and 75%. For the bass lines, it had a simple Triangle wave, and for drums it simply had a noise channel. Each channel could also receive a number of effect inputs, which included effects such as Vibrato, Decay rate, Portamento, Pitch Bend, etc. The VRC6 expansion controller included 2 additional Pulse waves with duty settings from 1/16th to 8/16th. It also included a Sawtooth generator, though often this was used for non-musical purposes.

The PureData patch has a total of 7 channels, with 4 individually controlled Pulse Waves of variable depth, 2 individually controlled Triangle waves, and a simple drum channel which uses a combination of noise and low-volume, aggressively enveloped sine waves. Additionally, the Pulse Wave channels all have a vibrato controller with 16 different frequencies, and all melodic channels can trigger a slide between notes. The second Triangle wave serves only to reinforce and double the bass line when the bass line hits the root of a chord, to reinforce the feeling of a chord progression.

The original hardware was not able to be faithfully recreated in PureData fully because of PureData's abstraction of floating point numbers, which removed the ability to reduce the bit-rate of a channel via simple mathematical functions. As I moved my focus away from pure recreation, I added additional features to the patch, including modern enveloping, and developing a ⅓rd-octave equalizer.

To connect PureData to Unity, I used a slightly modified version of Patrick Sebastien's "libpd4unity" plugin from GitHub. The only modifications which I needed to make were that the original plugin was slightly outdated. When the plugin was initially made, Unity used 32-bit architecture, and with the release of Unity 5, Unity moved to 64-bit architecture. This broke some dependencies in the plugin. To update the plugin to run on the newer 64-bit architecture of Unity 5, it was simply a matter of recompiling the dependencies and updating to 64-bit architecture.

The plugin allows for Unity to seamlessly load in PureData patches, and can send any message, value, or bang to a receive object in the loaded patch. During gameplay, Unity simply walks through each musical line, and sends the midi pitch value for the current line to PureData. To control volume, each line has both a base volume, which allowed me to fine-tune the relational volumes between each line, and a "volume multiplier", which is set by the position of the slider. Additionally, if there is a rest in a line, the volume is set to 0 until the next note begins. Because QCF stores pitch data in midi pitch, walking through the lines and sending the pitch values to PureData could be done with the raw generated data.

Appendix C: Manual

Figure 5: The ceiling showing 2 correct sliders, one low slider, and one high slider

Installation:

In order to install *The Audience of the Singular*, first download the application files. The files can be downloaded from https://optimalgoat.itch.io/aots

On Windows: Download either the .zip file and extract anywhere, or download *both* the .exe file and the _Data folder, and place them in the same location.

On Mac: Download either the .zip file or the .app file, and place them in any folder other than the "download" folder.

If there is a security notification(On Mac): Control-click the .app, select "Open", and then "Yes" when prompted.

Once the application is downloaded, open the application. Enter preferred settings, and click the "Play" button.

Gameplay:

In *The Audience of the Singular*, gameplay involves attempting to meet the arrangers chosen suggested mix as quickly as possible. If the player meets the suggested mix quickly, the audience becomes increasingly excited. At the beginning of each play session, the player selects the number of pieces which they want to play. At the end of the final piece, the player is assigned a score based on how excited the audience became over the course of the set. The player's highest score for each number of pieces played is stored on their computer, and they can attempt to beat their own high scores.

The player attempts to match the suggested mix by watching the ceiling as they play. The ceiling consists of 32 squares. These squares get clustered into 4 groups of 8 squares each. Each group is assigned a slider when the animation changes, which happens at the beginning of every phrase as well as each time the player has matched the suggested mix. These assignments are done randomly, and which squares correspond to which slider is information which is not immediately apparent to the player.

Each group of squares constantly compares its assigned slider value with the arrangers suggested mix, and changes colour based on that comparison. If the slider is below the suggested level, the group will turn blue. If the slider is above the suggested level, it will instead turn yellow. As the player approaches the suggested mix, these colours will become brighter.

An extremely basic flow of the gameplay loop can be understood in the following example:

- The player has a board where the Drums are at 3/4, the Bass is at half, the Rhythm is at maximum, and the melody is at half.
- The player looks at the ceiling and sees that it is mostly green, with a few bright blue squares.
    - The player can then infer that while 3 of the sliders are in the suggested position, one of them is slightly too low.
- The player chooses to manipulate the sliders from their left to their right until they have found the suggested slider to move.

- The player selects the drum slider, and moves it upwards. As they move it, suddenly the ceiling has yellow squares as well.
    - The player can now infer that the drum slider was one of the sliders which was in the correct position.
- The player repeats the process for the bass slider, and the same results occur.
- The player attempts to move the rhythm slider, but because it is already at maximum, it cannot move upwards, and therefore cannot be too low.
- The player selects the melody slider, and moves it upwards. As they move it, the blue squares disappear and turn green instead. Lights flash and smoke emitters activate.
    - The player has matched the suggested mix! The arranger will now select the next suggested mix.

While this version of playstyle works, it is inefficient, and the player will receive lesser rewards due to the time such a process would take. In order to maximize their speed, in order to maximize on the audience's simulated excitement, they would need to quickly deduce the more probably suggested mixes. A version of this can be understood in the following example:

- The player has a board where the Drums are at half, the Bass is at 0, the Rhythm is at maximum, and the melody is at ¾.
- The player looks at the ceiling and sees that it is about half green, with some bright blue and some dim yellow
    - They can immediately infer that two of the sliders are correct, one is slightly too low, and one is very high.
- The player makes the following assumptions based simply on the positions of the sliders:
    - The bass cannot be too high, as it is already at zero
    - The rhythm cannot be too low, as it is already at maximum
- Because the player has learned the tendencies of the arranger, they can assume the following:

    - The rhythm is likely to be too high, as the arranger generally tries to keep the melody above the rhythm.
    - The bass is likely to be too low, as the arranger generally likes to keep sliders from spending too much time at 0.
    - The drums may be too low, as the arranger prefers the drums and melody to be around the same level, and the ceiling is showing a very small amount of difference.
- The player then, rather than beginning the process by selecting the drum slider and waiting for a change, begins by moving the rhythm line down. As they do so, the yellow tiles become increasingly bright and eventually turn green
    - The player knew during this that they had picked the correct slider, as the hue of the yellow tiles was changing.
- The player then moves the bass up, and the ceiling again has yellow squares.
    - The player now knows that this slider, despite being likely to be in the 'wrong' spot, was still in the suggested spot.
- The player moves the drums up, as they are the players second choice for being too low. The ceiling turns green. Lights flash and smoke emitters activate.
    - The player has matched the suggested mix! The arranger will now select the next suggested mix.

This example provided the player with a far more difficult situation: there were 2 sliders out of position instead of one *and* the arranger chose a slightly divergent slider than expected. By learning the arranger's trends, the player is able to solve the more complicated mix in 3 steps instead of 4.

This learning process mirrors the process of learning to mix music, though in a controlled and artificial way. The player will, by playing the game and paying attention to trends, be able to eventually learn the likes and dislikes of their simulated audience.

The audience has three bars which indicate their reaction to the player's mix: Hype, Fatigue, and Excitement. Excitement and fatigue are linked, and change over time. When the arranger picks a new suggested mix, the excitement meter goes to its maximum level, based on the current excitement, and the fatigue goes to 0. While the player has not

matched the suggested mix, the fatigue meter slowly fills and the excitement meter slowly drains. When the player successfully matches the suggested mix, the audience gains hype equal to the amount of excitement.

When the hype meter reaches certain thresholds, the “level” increases. This level governs a wide array game aspects, including:

- Maximum excitement level
- Speed at which fatigue accumulates and hype drains
- Intensity of the lights
- Maximum level for each slider
- Specificity required for the suggested mix
- Number of audience members
- Willingness of the audience to dance
- Number of suggested levels that change each time a new suggested mix is chosen
- Amplitude of suggested changes each time a new suggested mix is chosen

At the end of the game, the hype is converted into an integer, and stored as a score. Each score is linked to the length of “set” which the player selected at the beginning, and the highest score that the player has achieved is stored locally.

A youtube video explaining this process is available at http://at.sfu.ca/NXyXZY.