Interference: Adapting Player-Music Interaction in Games to a Live Performance Context

Matthew Wang
Princeton University
Princeton, NJ 08544
mjw7@princeton.edu

ABSTRACT

Interference is a multiplayer music game and generative music system, which is implemented as a Javascript web application and designed for live performance. It is based on the potential for dynamic music generation that exists in video games through player-music interaction. It uses a competitive multiplayer form to sustain a feedback loop in which players construct and change the music at a fine scale, while the music in turn informs players of the game state, affecting how they continue to play and therefore change the music. The design of *Interference* must also manage conflicts between games and music as contrasting media, such as presentation, length, and complexity, in order to create both a game that is engaging for its players and a musical performance that is compelling to its audience. Towards this objective, it combines elements of games that do not traditionally exist in music, such as an explicit goal-oriented structure, with features that serve strictly musical, performative purposes, allowing players to act simultaneously as performers. To support this design, it utilizes several existing web technologies to achieve tight synchronization, changeable sound synthesis, and networked interaction between players.

1. INTRODUCTION

A crucial consideration in the development of video games and especially in the creation of their music is how players interact with the music and sound of games. Due to the inherently dynamic nature of games, their music must be changeable based on how players can progress through them. For most games, this means that music starts in response to a trigger – possibly on startup, entering a new area, or encountering a certain character – and continue for its set length or indefinitely until another trigger. The player, having control over each trigger either directly or indirectly, therefore controls the music. Inversely, the music of games influences how players act. It can inform the player of a change in the state of the game, prompting them to react, or exert some emotional effect on the player, affecting their decisions or perceptions going forward.

This combination of player influence on the progression of music and musical influence on the actions of players results in a feedback system. Player action determines the progression of the music, which affects further player action in turn. But for a majority of games, the music-making potential of this feedback system goes unrealized. This is largely unsurprising in games that do not place significant focus on music as a game element. In these games, non-musical goals are the primary concern and more complex player-music interaction, if implemented, would likely obscure those goals. However, even most music-oriented games make little use of this feedback system. Take for example the archetypal "music game" *Guitar Hero* [2]. Although it presents as a music-making game, in which the player acts as a performer, the player actually has very limited control over the music. They are only able to play or not play predetermined notes – controlling the playback of music, but not its content. Michael Liebe describes this form of music, which is typical of music games, as proactive music, which mostly occurs independently of player input and demands some response from the player, as opposed to reactive music, which reacts directly to player action, and linear music, which occurs independently of direct player input and does not demand a response [5].

With *Interference*, I aim to create a game and performable generative music system that takes advantage of player-music feedback and features music that is simultaneously proactive and reactive. From a design standpoint, this requires an understanding of approaches to sustaining such a feedback system, methods for the construction of a compelling game that uses said feedback system, and any concerns a performance context may introduce. Following a discussion of these points, I will explain the core components of the technical implementation of *Interference* as well as its gameplay and performance.

2. DESIGN

2.1 Player-Music Feedback

The main obstacle to sustaining a consistent and meaningful feedback relation between player and music is the inherent incompatibility of reactive music with proactive music. Reactive music struggles to influence player action in the same way as proactive music because players tend to interpret reactive music as their own action or as a commentary of their action rather than as an external force that demands reaction. Conversely, if the music reacts more indirectly to player action such that players feel they must respond to it, it quickly ceases to feel reactive as players become be less able to purposefully influence it.

One solution to this problem is to allow the player to directly generate the music but then introduce a level of abstraction or error into that generated music that retains its reactive identity while still demanding a response from the player. An example of this sort of feedback system is *Zero Waste*, a game-like musical piece for a

Web Audio Conference WAC-2019, December 4–6, 2019, Trondheim, Norway.

sight-reading pianist and a computer, which has the computer present the performer with a few measures of random music to sight-read before transcribing an attempt by the pianist to play it, and presenting the transcription back again [8]. This system achieves a form of feedback that could exist in-game, but it relies on player and system error and trends. As a result, there is a somewhat deterministic quality to the process, and furthermore, change tends to occur gradually in such a system.

For this project, I sought to create a more dynamic and variable system. Therefore, to overcome this obstacle of sustaining a feedback system, rather than use error to create variation in a human-computer feedback system, I chose to design *Interference* as a multiplayer game. With multiple players, each player can contribute in a direct and transparent way to the composite generated music, but each player must also react to the contributions of every other player. This maintains both reactive and proactive features in the music.

2.2 Competitive Gameplay

Such a multiplayer game could be implemented as either a cooperative or competitive system, but I chose to create a competitive system for several reasons. First, competition, more automatically than cooperation, engages players and encourages participation because it introduces conflict and challenge without a need for high complexity or technicality. Second, concerning use in a performance context, a competitive system requires only a minimal level of skill and knowledge in the mechanics of the game in order for a performance to progress and, importantly, end.

Last, the combination of proactive and reactive musical elements in a multiplayer environment quickly becomes chaotic and complex. This complicates the implementation and balance of cooperative tasks, which require a non-player system to challenge but not overwhelm players, whereas a competitive system is largely self-balancing even in a highly complex environment. Regardless of the complexity of the system, each player starts with equal means and information and is only rewarded or limited by their own ability to use their resources in comparison to their competitors. In this way, while a player could choose to ignore proactive musical information, they would put themselves at a disadvantage to more reactive players.

2.3 Performance Context

In contrast to the competitive nature of *Interference* is its design as a performance work. While the players compete, they must also cooperate as performers and therefore may sometimes choose to take performative actions rather than actions that are competitively advantageous. Therefore, a design goal was to have performance and competitive motivations overlap as much as possible, such that actions which are advantageous in the game are also musically engaging.

An important consideration towards achieving this goal is how player action maps to sound. Ideally, game-motivated player action should correspond to musically functional sound. To the determination of strong mappings as such, attention to the time scale of game events is especially important. With some exceptions, game events that occur only occasionally best map to more significant changes and large-scale shifts in the music while changes that occur frequently map well to more subtle changes in the music. In particular, if the musical result of an action that occurs frequently in the game is too extreme, it can create conflict between the player and the performer roles. Karen Collins' "Ten Approaches to Variability in Game Music" from *Game Sound* is a particularly useful resource in the consideration of which elements of music in games can be effectively dynamically altered, although the actual mapping of game events to these elements is largely a matter of experimentation [1].

Interference specifically achieves some harmony between the roles of player and performer in that large modal shifts in the game correspond to paradigmatic changes in the musical texture while the most frequent game events change the music only incrementally. And perhaps most successfully, one of the strongest tactics – a technique I call "leading the sequencer step," discussed in Section 4 – results in appropriately striking yet simple musical figures. That said, *Interference* is somewhat lacking in the range of performative expression it gives its players due to its limited use of variable sound synthesis and the relatively small amount of influence a single performer has over the composite music.

A final performance concern, aside from the harmonization of player and performer roles, is the practical execution of these roles, which are challenging for a single person to focus on simultaneously. *Interference* uses its visuals to enable its players to execute both roles. By corresponding to both game action and musical change as closely as possible, visual elements help link the game and music. Visual elements that simultaneously represent a game object and appear to produce sound allow players to act deliberately as performers. Additionally, these visuals can enhance the experience of the audience, which led to the choice in performance to display players' screens on external monitors facing the audience (see Figure 1).

Figure 1. Setup of a five-player performance.

3. IMPLEMENTATION

3.1 Networked Game Interaction

Crucially, *Interference* is a networked game, as information about game and player states must be shared across all players. For networking game information and input between players, I used Lance, a Node.js based server and client-side library intended for multiplayer web-based games [7]. It additionally includes basic game and physics engines and synchronization strategies to handle latency. Lance does not provide for the more flexible sound synthesis or the precise rhythmic synchronization *Interference* requires but is extremely useful for essentially all other aspects of the project, and additional modules and libraries are easily incorporated.

3.2 Synchronization

Due to both the musical focus of this project and the presence of input latency in a networked game context, which exacerbates any synchronization issues for performers, implementing precise and reliable rhythmic synchronization was especially important. For this synchronization, I used Collective Soundworks' `sync`, which provides consistent synchronization with the minor condition that players may need to wait some time for their sequencers to synchronize upon connection [3, 4].

3.3 Sound Synthesis

Finally, effectively controlling the music of *Interference* requires flexible sound synthesis. Adequately flexible sound synthesis in a game context allows for virtually any mapping of game variables to sound. For this purpose, I ultimately chose to use Tone.js, an audio framework built on the Web Audio API [6]. Tone.js has some notable limitations, such as a lack of polyphony on noise-based instruments, but as a relatively mature and widely used web audio framework, its ease of use was worth any of its inflexibilities.

4. GAMEPLAY

To balance the complex fine-scale variation of the musical and visual elements of *Interference*, the game itself is relatively simple if rather abstract. The game space consists of a series of 32 by 18 colored grids positioned in a horizontal line. Each grid represents one player's territory, their initial field of view, and a set of three step-sequencers layered on top of each other (see Figure 2). Each of these three step-sequencers runs constantly at differing rates from one another. The horizontal axis represents the sequencer time steps and the vertical axis represents the pitch of sequenced notes. Each player begins as one of five color palettes, which correspond to various harmonic sets for the sequencer pitches. Each player's goal is to convert every other player to their palette.

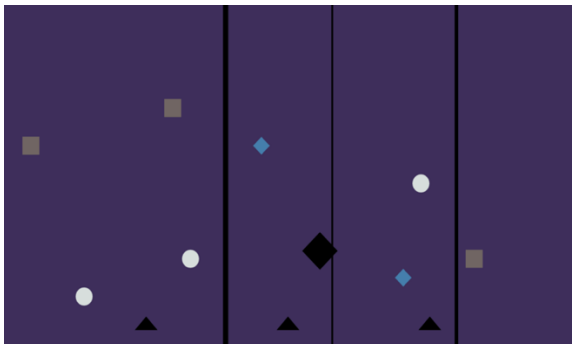

Figure 2. The view of a single player's grid during a build phase with a diamond-shaped ball in black, notes of each shape, and playheads for each sequencer. The black triangles at the bottom of the grid indicate how many notes the player has left to place.

With the exception of an outro section, the game consists of two states that alternate over its course: the "build" phase and the "fight" phase. See Figure 3 below, which provides an outline of the overall game progression structure and the mechanical and musical characteristics that define each type of phase.

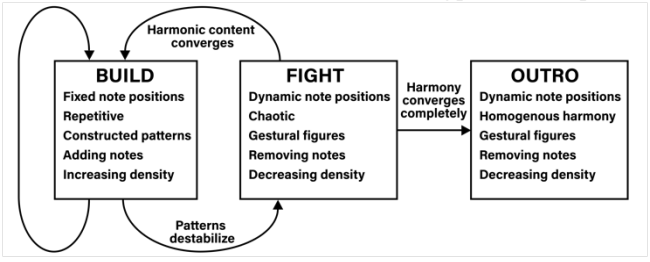

Figure 3. The progression of *Interference*. The alternation of build and fight phases creates a cycle between varying levels of density and stability as the overall harmonic content gradually converges.

The game begins in a build phase, during which players build sequences of notes. To start, a ball object spawns with a randomized position and velocity and moves throughout the space, bouncing off its boundaries. Players can then "hit" the ball as it passes through their area to place a note in their sequencer at the position of the ball. The shape and sound of the placed note depend on the shape of the ball, which can be a circle, a diamond, or a square. After players have collectively hit the ball enough times, it breaks. The player who broke the ball then has the option to start another build phase or progress the game into a fight phase.

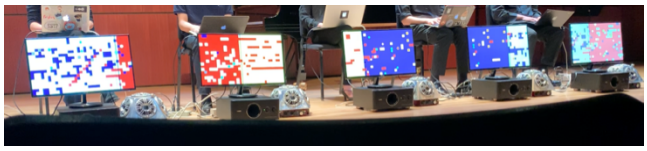

Figure 4. Players' screens during a fight phase with their views displaced from their starting position.

During a fight phase, players can move their view and placed notes as a single rigid structure through the entire game space, wrapping across boundaries (see Figure 4). Players convert cells of the grids to their color when a sequencer plays their notes on those cells, resulting in the especially effective tactic of leading each step of a sequencer playhead, such that a note plays on every step and converts a line of cells. Players can remove each other's notes by forcing collisions with their own notes. A rock-paper-scissors system using the three note-shapes determines which note to remove upon collision.

After enough notes have been removed or players have made enough inputs to force progression, the fight phase ends and the piece transitions back into a build phase. At this point, players and notes reset to their original positions and if a player's territory has been mostly overtaken by other colors, their entire territory and their notes convert to the now most prominent color. The game ends when all players have the same color palette or after some amount of time chosen beforehand, at which point all players are converted to the dominant color (see Figure 5).

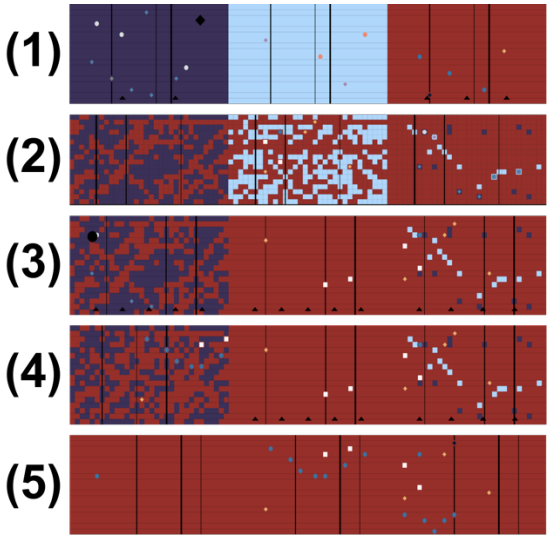

Figure 5. Progression of a three-player game. (1) First build phase. Each player has a color. (2) End of the first fight phase. The red player has converted much of the field. (3) Second build phase. The light blue player has been fully converted to red upon the transition from fight phase to build phase. (4) End of the second fight phase. The red players have converted even more of the remaining field. (5) Outro phase. The purple player has been converted to red and the game has ended.

Aside from the basic rules and progression of the game, the gameplay also intersects with the music in several ways. For example, because each player's audio output only sonifies the immediate contents of their territory, they can use their audio to identify when another player is converting their territory, assuming players use spatially separate audio output systems, as with the Princeton Laptop Orchestra's hemispherical speakers. Players can also use the music to identify the state of the game, such as the current game phase or even the dominant color palette. Players can, of course, identify these elements visually but not without a significant time commitment due to the limited scope of visual information at any given moment, so those who can react effectively to the music gain an advantage.

5. PERFORMANCE

Interference also features several strictly performative features, which generally cater to musical elements that were otherwise difficult to incorporate into the game. The outro section mentioned previously is one such case as it begins only once the actual game has ended. During the outro section, all players have the same color palette and can move freely as in a fight phase, but instead of removing each other's notes, players can input a command to remove a random note from the whole game. The purpose of the outro section is strictly musical in that it allows the performers to play freely in the harmonic progression of the final color palette and slowly time a fade-out by removal of notes to circumvent an otherwise musically abrupt ending to the game. Another performative feature is a control to progress the harmony, which during a fight phase also controls progression back into a build phase, but otherwise strictly serves a musical purpose.

6. CONCLUSION

Interference explores a form of feedback-based music generation that game environments provide but that often goes unused due to limitations games typically impose on their music. Rarely do games allow music to make heavy demands of players and when they do, as in music and rhythm games, the player generally has little control over the generation of the music. By creating a multiplayer system in which players interfere and intermingle with one another's musical and game choices, *Interference* allows its players to generate its music while simultaneously allowing the music to make demands of its players.

The concepts behind this project open up many opportunities for future development. While *Interference* features a high level of fine-scale variability and no large precomposed parts, which are more typical in games, these ideas could potentially apply to any scale of musical content. Additionally, many features of *Interference*, such as the visuals, exist mostly to make the game more accessible to players and audiences and are not necessary to the core concept of the game. A strictly audio-based game could be possible (though much more difficult to play), and as previously discussed, a cooperative game could operate under similar principles through careful design. The generative possibilities of multiplayer music games are largely unexplored, and *Interference* is only a basic proof of their potential.

7. LINKS

Interference is playable at https://interference.herokuapp.com/. Server limitations may affect game performance.

Source code and a brief description of controls is available at https://github.com/mattmora/interference.

Video and audio recordings of the premiere performance of *Interference* by the Princeton Laptop Orchestra is available at https://www.youtube.com/watch?v=C-5P3hXuGfs.

8. ACKNOWLEDGMENTS

Thanks to Jeff Snyder for advising the development of this project, to the Princeton Laptop Orchestra for their assistance in testing, rehearsing, and performing it, and to my friends in the Princeton Pianists Ensemble for taking part in a second performance.

9. REFERENCES

[1] Collins, K. 2008. *Game Sound: An Introduction to the History, Theory, and Practice of Video Game Music and Sound Design*. MIT Press, Cambridge, MA and London.

[2] Harmonix. 2005. *Guitar Hero*. Game [PlayStation 2]. RedOctane, Mountain View, California, USA.

[3] Lambert, J.P. sync. Github Repository. Retrieved from https://github.com/collective-soundworks/sync, last checked 11 March 2019.

[4] Lambert, J.P., Robaszkiewicz S., and Schnell, N. 2016. Synchronisation for Distributed Audio Rendering over Heterogeneous Devices, in HTML5. In *Proceedings of the Web Audio Conference (WAC)*. Atlanta, USA. pp 6–11. URI= http://hdl.handle.net/1853/54598.

[5] Liebe, M. 2013. Interactivity and Music in Computer Games. In *Music and Game: Perspectives on a Popular Alliance*, by Peter Moormann. Springer VS, Berlin. pp 41-62.

[6] Mann, Y. Tone.js. Github Repository. Retrieved from https://github.com/Tonejs/Tone.js, last checked 11 March 2019.

[7] Weiss, G. Lance. Github Repository. Retrieved from https://github.com/lance-gg/lance, last checked 9 March 2019.

[8] Whyte, D., Didkovsky, N., and Hutzler S. 2018. Zero Waste: Mapping the Evolution of the Iterative Sight-Reading of a Piano Score. *Music Theory Spectrum* vol. 40, 2, pp 302–313. DOI= https://doi.org/10.1093/mts/mty019.