SoundCraft: Transducing StarCraft 2

Mark Cerqueira
Smule, Inc.
577 College Avenue
Palo Alto, California
mark@smule.com

Spencer Salazar
Center for Computer Research in Music and Acoustics (CCRMA)
Stanford University
spencer@ccrma.stanford.edu

Ge Wang
Center for Computer Research in Music and Acoustics (CCRMA)
Stanford University
ge@ccrma.stanford.edu

Figure 1: A SoundCraft performance in progress.

ABSTRACT

SoundCraft is a framework that enables real-time data gathering from a *StarCraft 2* game to external software applications, allowing for musical interpretation of the game's internal structure and strategies in novel ways. While players battle each other for victory within the game world, a custom *StarCraft 2* map collects and writes out data about players' decision-making, performance, and current focus on the map. This data is parsed and transmitted over Open Sound Control (OSC) [9] in real-time, becoming the source for the soundscape that accompanies the player's game. Using SoundCraft, we have composed a musical work for two *StarCraft 2* players, entitled *GG Music*. This paper details the technical and aesthetic development of SoundCraft, including data collection and sonic mapping.

Keywords

interactive sonification, interactive game music, StarCraft 2

1. INTRODUCTION

Games are a medium of expression. They are inherently interactive, social, and expressive. They can also be competitive and nuanced. Take those layers of nuance and put sound to them, and one perceives the game in a different way.

The world has become more interconnected via technology, and video games have become more social as well. Massive communities have built themselves in and around video games fostering cooperation, collaboration, exploration, expression, and many times competition among its members. In its conception, SoundCraft seeks to unravel the layers of nuance present in a widely popular game that centers around conflict between players and reinterpret those layers in real-time to build an accompanying musical experience.

In its realization, SoundCraft seeks to open up a complex game engine to create opportunities for creative and novel interpretations of the game.

2. BACKGROUND

SoundCraft centers around the widely popular RTS or "real-time strategy" video game series *StarCraft 2* by Blizzard Entertainment. In *StarCraft 2*, players control one of three races: the Terran, humans who were exiled from Earth long ago; the Zerg, an insect-like alien species mindlessly bent on overrunning and assimilating other species; and the Protoss, a humanoid species with highly advanced technology and great psionic abilities seeking to annihilate the Zerg [1]. The game includes a story-driven campaign mode, featuring cinematic story-telling, that explores the struggles of each of the races in the universe. Aside from the campaign, players can play custom maps designed by community members or battle against other players over the Internet.

A competitive scene has even built itself around *StarCraft* with leagues and tournaments taking place all over

[1] http://us.battle.net/sc2/en/game/race/

NIME'13, May 27 – 30, 2013, KAIST, Daejeon, Korea.

the world drawing crowds of spectators to cheer for their favorite professional gamer or team. Fans watch the game as a spectator sport from the perspective of an all-seeing director/observer who decides what action is shown, and commentary is normally provided by casters who have deep knowledge of the game and strategies.

During a game, players see the action from a top-down perspective and only have vision of the map where they have units or buildings present. Players switch between building up their economy by constructing workers that collect resources and building an army to do damage to their enemy. A player who is able to build a strong economy and the proper composition of units that counter their opponent's units is in an advantageous position. The game ends when a player destroys the other player's structures, or the other player surrenders.

3. RELATED WORK

SoundCraft's use of a video game as an interface for producing music is directly influenced by many other works and research in the computer music community.

In the 2008 paper *Composing for Laptop Orchestras*, founding members of the Princeton Laptop Orchestra (PLOrk) discuss the possibilities of using games in computer music environments as well as some of the game-centered pieces they have developed. One early and relevant piece to SoundCraft is Scott Smallwood's *The Future of Fun (1983)*, which has each performer playing a different game from the Multiple Arcade Machine Emulator (MAME). Performers pick a game from MAME, which features games from the early 1980s arcade-game era, such as *Pac-Man*, *Defender*, and *Donkey Kong*, and play the game for a specified amount of time. The sounds each game produces become the piece, which while different from game to game, contributes to a soundscape of low-resolution synthetic sounds common to that era [6].

Rob Hamilton of Stanford University's Center for Computer Music Research and Acoustics (CCRMA) developed a modified version of the popular first-person shooter (FPS) *Quake 3* to enable bi-directional communication between a game and external audio servers via OSC. Dubbed *q3osc*, the modified game allowed players to connect with each other over a local area network or internet and control an avatar that could fire projectiles. The data for these projectiles is shared via OSC and in one of the mapping schemes Hamilton explores, the collisions of projectiles and environment surfaces triggered impulses through a resonant filter in a ChucK program [3].

Hamilton continued his work with game engines, modifying the Unreal Developer's Kit (UDK) to add OSC functionality, creating the UDKOSC framework [4]. This framework was used by Charles Verron and George Drettakis to explore plausible and efficient implementations of sound synthesis for rain, fire, and wind that could be used for video games. Verron and Dettakis created different, fundamental sound atom units that could be combined to form these sounds. To test their implementation they created an interactive scene using the UDK game engine for graphics rendering, an external Max library for sound rendering, and the UDKOSC library to communicate between them. The scene also supported spatial distribution of sound sources for localized sources (fire) and diffuse sounds (rain and wind) [7].

4. WHY STARCRAFT 2?

Of fundamental importance to the sonification of a computer game is the gameplay itself. In terms of gameplay, a run-and-gun first-person shooter (FPS) game emphasizes the perspective of a single actor within the game and his or her direct interactions with the game world and other players. As seen in Hamilton's work, such gameplay is readily sonified with sound sources distributed throughout the game world, spatialized from the perspective of the player. Specific in-game actions, such as flipping a switch or shooting a projectile, are mapped directly to a sonic result. Real-time strategy games differ significantly in terms of gameplay, in that the perspective is an overview of the current game state and interactions with the game world are indirect. The player gives orders to large groups of worker and soldier units in the game, which carry out those orders based on simple artificial intelligence algorithms.

Consider moving a unit or group of units from point A to point B. In a FPS game, a player has control over a single character and simply moves that character from A to B. In *StarCraft 2*, a player issues the move command to a select group of units and chooses the destination. A pathfinding algorithm takes over and handles getting units from A to B in the shortest time possible. While Blizzard has not publicly discussed its pathfinding technique, many community members inferred that *StarCraft 2* uses techniques based on sophisticated flocking AI [2], which attempts to mimic the movements of a flock of birds or a school of fish. Craig Reynolds explored and implemented one of the earlier models for flock simulation for Boids, a generic flocking creature. Flocking consists of a combination of steering behaviors - the most fundamental behaviors that allow a character to navigate their surroundings in a *lifelike* manner [5]. Such artificial intelligence behaviors have been employed for musical purposes in Hongchan Choi's *Lush*, an "organic" musical sequencer program in which groups of boid creatures, controlled by flocking behavior, activate musical patterns as they flit about the screen [2]. SoundCraft, by harnessing artificial intelligence behaviors already present in *StarCraft 2*, presents similar opportunities to orchestrate musical works.

Figure 2: Harvesters mining minerals and vespene gas, which triggers musical events.

In addition to the complex algorithms used for unit movement, units themselves automatically behave in ways that can be musically interesting. Consider a harvester unit that mines resources, like a mineral node, and returns it to the foundational structure where it then becomes available for use by the player (Figure 2). There is a complex set of rules that govern this entire process, which the player may be unaware of, as it is all done automatically. The entire process begins with a harvester unit accelerating to full speed as it moves towards the node, decelerating to a stop as it approaches the node. Once stopped at the node, the harvester mines the node for exactly 2.768 seconds, then pauses for

[2]http://www.teamliquid.net/forum/viewmessage.php?topic_id=132171

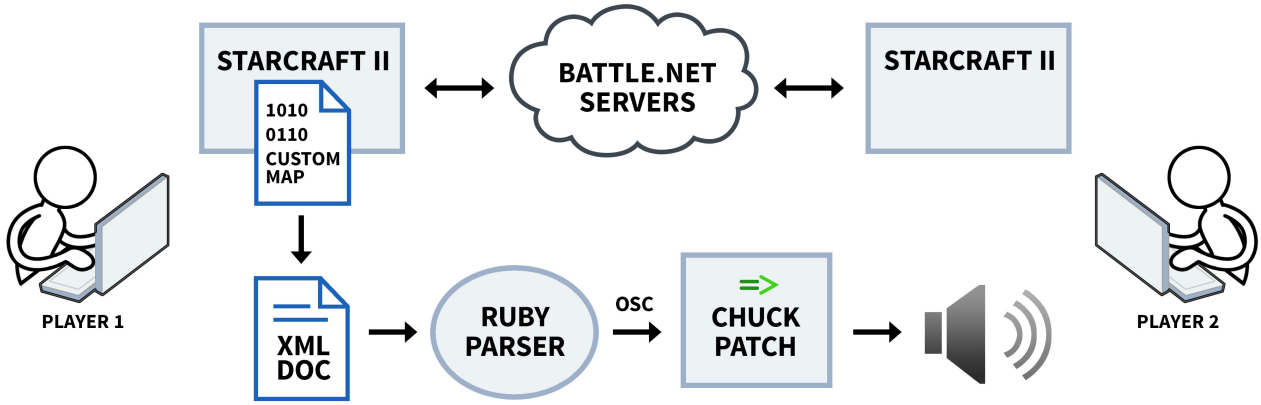

Figure 3: Flow of data among players and between the game space and auditory space.

0.5 seconds, before it accelerates to full speed away from the resource node towards the foundational structure. As it approaches the structure to deposit the collected minerals, it decelerates to a stop [3]. While monitoring one worker harvesting one node may seem only marginally interesting, when you scale up to the eight mineral nodes a typical base will have, with the optimal three workers per node, and then include vespene gas gathering which operates on a slightly different timing mechanism, a far more interesting scenario is created. These workers become members of an autonomous multi-agent system, which has been explored for music composition in the past. Researchers from Zurich University of the Art's Institute for Computer Music and Sound Technology developed the Interactive Swarm Orchestra (ISO) that supported the creation of multi-agent systems consisting of point-like members possessing simple behaviors [1].

In *StarCraft 2* a player is able to give different units different orders. This allows for parallelization, which is not available in games where a single player controls a single character by giving it a set of instructions that are processed in a serial fashion. Not only can different actions be issued as fast as a player can input them, a player can also queue up orders, creating a complex set of actions to execute automatically. For example, a group of units can be told in one string of commands to board a dropship that will then transport the units across the map, drop those units off, and then have the units move to another point before attacking potential enemies. While the units are executing those orders, another set of units can be instructed to move to potential enemy bases, attacking any enemies found along the way. The game itself can be extremely high-action at times, even though the player at that precise moment may not even be watching as his units simply execute on orders given several moments before.

These dimensions of *StarCraft 2*'s gameplay can be further expanded by customizing the maps on which players do battle. Blizzard provides a tool just for this purpose, the Galaxy Map Editor. The design of a map greatly influences which strategies will be successful and the general flow of the game. With the Galaxy Map Editor, a map creator can decide what types of terrain features players must overcome to fight their enemy, what additional territory is available to harvest economic resources from, and many other map design aspects. This diversity of potential gameplay translates into a variety of player actions, providing a rich field for sonification.

The three different races of the *StarCraft* universe offer another interesting opportunity for musical expression. Not only does each race have a different look and feel as well as a set of different units with distinct abilities and properties, but the strategies employed by each race vary as well. For example, all Zerg units come from the same pool of resources: larva. Since a strong economy is key to winning games, a Zerg player will want to spend as much larva as they can on building workers that collect resources. Only when a threat is incoming will a Zerg player want to spend larva on the necessary fighting units. The other two races, Terran and Protoss, do not share this constraint of sharing economy-building and army-building. Thus, even if a simplistic mapping were created where all worker units and all fighting units were sonified equivalently across all races, the audible result would vary from one race to another as *when* these units are built varies.

5. SYSTEM OVERVIEW

SoundCraft consists of a set of utilities that work in tandem to get data from a *StarCraft* game to OSC clients that can creatively express that data. A diagram of the system is shown in Figure 3. The main components include a custom *StarCraft 2* map developed with Blizzard's publicly available map-making tool, and a game data output parser and OSC dispatcher developed in Ruby.

5.1 Creating the Custom Map

An existing *StarCraft 2* map, Bel'Shir Beach, shown in Figure 4, created by community member LSPrime [4] , was modified to collect various data in real-time during a game using *triggers*. *Triggers* can be thought of as instructions that execute after an event occurs. Events that cause triggers to run can be occurrences in the game (e.g. a unit is produced) or can be fired off periodically (e.g. every 2 seconds) [5]. For example, one particular trigger used extensively in SoundCraft collects game state and writes it to a set of XML files. The map created while developing SoundCraft is for two players battling against each other, but the triggers can be shared to any other map with any number of players or

[3]wiki.teamliquid.net/starcraft2/Mining_Minerals

[4]http://wiki.teamliquid.net/starcraft2/Bel'Shir_Beach

[5]http://us.battle.net/sc2/en/game/maps-and-mods/tutorials/trigger/

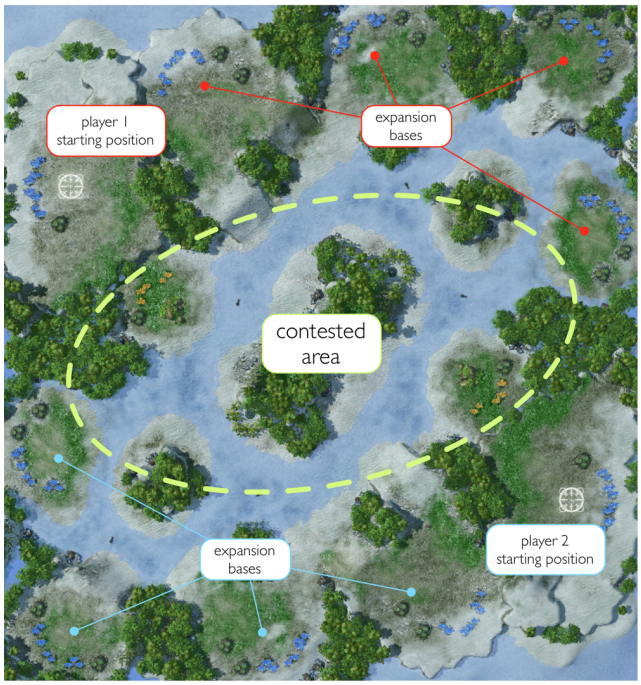

Figure 4: Bel'shir Beach, with points of interest highlighted.

custom games that feature custom objectives and story elements.

Data collected during the game focuses on events as small as an ability being used by a single unit to more macroscopic events such as the economic status of a player.

Tracked data includes:

- The player's race (i.e. Terran, Protoss, or Zerg)
- Coordinates where a player is currently focused on the map
- Composition of the player's standing army
- Units and structures being produced by the player and their current build progress
- The player's effective actions per minute (eAPM)
- Various metrics about a player's economic health including mineral and vespene gas collection rate as well as harvesters built and lost
- Abilities used by units (e.g. Chrono Boost, Spawn Larva, Calldown MULE)
- Total number of enemy units destroyed and enemy structures razed
- A notification when a player surrenders or leaves the game
- Any messages a user types in the in-game chat
- A notification when a player begins adding production buildings with periodic updates on building progress
- The player's general *score* that combines many game-related measurements to produce a single number that is normally a good indicator of performance (this is typically only shown after a match has ended)

5.2 Aesthetics of Map Design

While developing SoundCraft, we chose to use an existing map so as to focus on building and testing the framework, but map design plays an integral role in providing the visual setting for the aesthetic of a piece. While users of SoundCraft may opt to develop their own systems for visually interpreting the data being generated, we feel the game itself provides an interesting domain for visual experimentation. Tilesets are fundamental in creating maps and encompass the background elements and terrain available for use. *StarCraft 2* features twenty tilesets (named after planets in the *StarCraft* universe) that encompass different and varied environments, including volcanic, jungle, and desert worlds; destroyed cities; and even indoor industrial environments [6]. Bel'Shir Beach, shown in (Figure 4), uses the Jungle tileset.

Tilesets provide the background for a plethora of customization options for a map in the map-editing software. Terrain itself can be modified to include cliffs, valleys, canyons, bodies of water (or toxic waste if that fits better), roads, and various types of foliage that complement the tileset being used (e.g. trees in the Ice tileset will be frozen or have snow on them, while trees in the Jungle tileset will be in bloom and filled with leaves) [7]. In addition to modifying the environment, many other elements can be added to the map. To list a few: buildings of various types, fences, fossils, birds, fish, rocks, trains, and even non-playable characters that players can interact with can be added to a map. Even the weather can be manipulated to include rain, snow, lightning, sporadic volcanic flares, and clouds. If the editor does not offer something you are looking for, you can construct it yourself, but it is more likely that a member of the active map-making and modding community has already built it!

With this flexibility in building a map, a unique environment tailored to the particular story one wants to tell can be constructed. Perhaps instead of designing a map for two players to battle against each other, you could design a map featuring cinematic storytelling that has a player controlling a single unit as it traverses diverse environments questing for some mythological item, meeting allies and enemies along the way. Users of SoundCraft should certainly take this powerful point of customization into consideration when designing a piece.

5.3 Parsing and Sending Game Data

A Ruby script monitors the directories that contain these XML files using a file system monitor, guard/listen [8]. When the files in the monitored directory are updated, the data is parsed out of the XML files using Nokogiri [9] and that data is sent to any listening OSC clients on designated IP and port addresses. Via IP address designation, this data can be kept local to the player who is producing the data or it can be shared with all other players. Unfortunately, this is the only *legitimate* way to get data out of *StarCraft 2* as the game is a closed system that does not allow customization outside of the functionality allowed by Blizzard. We found the latency between a trigger firing and an OSC client receiving the relevant message to be acceptable given we had fine control over how often our script polls the file system for XML file updates. With operating system-specific file system polling adapters, the overhead of writing to a file, catching that event, and parsing the data produced low latency (i.e. on the order of 50 milliseconds) and acceptable CPU usage.

6. SONIFICATION OF GAME DATA

Note that the aim of SoundCraft is not to define how this game data should be mapped, but to open communication

[6]http://starcraft.wikia.com/wiki/Tileset
[7]http://us.battle.net/sc2/en/game/maps-and-mods/tutorials/terrain/
[8]https://github.com/guard/listen
[9]http://http://nokogiri.org/

between the game and any external modules that will respond to actions and events in the game. Our interpretation of the data should certainly not be viewed as the only way to interpret the data, but merely as a single instance of how data can be interpreted.

The gameplay of *StarCraft 2* is divided into two primary activities, developing the economy of a player's base, and producing offensive units to disrupt and destroy the opponent's own offense and base. Secondary activities include constructing additional production facilities, allowing offensive units to be trained faster, and researching technology upgrades, which improves the destructive effectiveness of all the player's offensive units.

Given this framework, we have chosen to sonify *StarCraft 2* gameplay as follows, in the context of a musical work titled *GG Music*. As a harmonic foundation, mineral and vespene gas mining (both economic activities) are realized aurally as FM plops. Each time a worker unit returns a mined resource to the foundation structure, a plop sounds. As a player builds his or her economy, more and more workers are trained and assigned to mining, gradually increasing the density of plops. In typical *StarCraft 2* strategies, vespene gas mining does not being until 1-2 minutes into the game; these events are sonified in a higher register than mineral mining, lending to a sense of musical progression.

The production facilities in which offensive units are trained are musically manifested as a low bass drone. As a player's economy grows, more production facilities can be built, enabling a larger and larger offensive force. Additional production facilities result in the bass drone inharmonically rising in frequency, increasing the musical tension as the two players build up their armies. Near the end of any given game of *StarCraft 2*, each player may have a dozen or more production buildings; at this point the bass drone is high enough to be more of a treble drone, which is musically much more uncomfortable.

Offensive units are represented musically as arpeggios of enveloped sawtooth oscillators. The number of notes, range of registers, and melodic contour of the arpeggio are augmented as the size of a player's army increases. Moreover, distinctive unit types are represented by different arpeggio patterns. For example, a standard Terran army often comprises Marines and Marauders. This army would be auralized in our system by two distinct arpeggio patterns. The more unit types compose a player's army, the greater the complexity of the resulting set of arpeggios.

Lastly, technology upgrades, which augment an army's offensive power, are not directly represented musically, but rather influence the parameters of other sonifications in the system. There are dozens of possible upgrades; however some upgrades cannot be researched until other upgrades have been unlocked or specific upgrade facilities have been constructed, creating a "technology tree" within *StarCraft 2's* gameplay. As a player accumulates different upgrades, the character of the bass drone and army arpeggios are tweaked. The modulation index of the FM bass drone is bumped for every upgrade; initially, this merely augments the presence of the drone. In the end game, however, the over-modulated bass drone fits uncomfortably in the mix, heightening the tension musically as a player maximizes his or her upgrades. Similarly, upgrades lengthen the envelope of the arpeggios, raising it from a short chirp to an elongated tone.

7. CONCLUSION

The sonic mappings explored here are but one of many. Through its unique, varied gameplay and large-scale simulation of artificially intelligent units, *StarCraft 2* offers interesting opportunities for developing complex computer music works. SoundCraft, via a custom map, a Ruby parser, and a ChucK-based [8] auralization, has enabled us to realize one such work: *GG Music*. SoundCraft furthermore seeks to provide a rich platform for diversity and expression through *StarCraft 2*. To facilitate this, the source code for SoundCraft as well as documentation and archived media is maintained on GitHub [10].

8. ACKNOWLEDGMENTS

Many thanks to the *StarCraft 2* mapmakers, modders, and enthusiasts at SC2Mapster for their support in building our custom map with special thanks to members: Ryan Schutter, deathtorn, Ahli634, zeldarules28, and willuwontu. We would also like to thank Tasteless and Artosis for fanning our love for the *One True Game*, Alyce Tzue for figure design, and Charles Pence for LaTeX support.

9. REFERENCES

[1] D. Bisig, M. Neukom, and J. Flury. Interactive Swarm Orchestra A Generic Programming Environment for Swarm Based Computer Music. *Proceedings of the International Computer Music Association Conference*, 2008.

[2] H. Choi and G. Wang. LUSH: An organic eco+ music system. In *Proceedings of the 2010 Conference on New Interfaces for Musical Expression (NIME 2010)*, 2010.

[3] R. Hamilton. q3osc: or How I Learned to Stop Worrying and Love the Game. *Proceedings of the International Computer Music Association Conference*, 2008.

[4] R. Hamilton. UDKOSC: An immersive musical environment. In *Proceedings of the International Computer Music Conference*, 2011.

[5] C. Reynolds. Steering Behaviors for Autonomous Characters. *Proceedings of Game Developers Conference*, pages 763–782, Spring 1999.

[6] S. Smallwood, D. Trueman, P. R. Cook, and G. Wang. Composing for Laptop Orchestra. *Computer Music Journal*, 32(1):9–25, Spring 2008.

[7] C. Verron and G. Drettakis. Procedural audio modeling for particle-based environmental effects. *Proceedings of the 133rd AES Convention*, 2012.

[8] G. Wang. *The ChucK Audio Programming Language: An Strongly-timed and On-the-fly Environ/mentality.* PhD thesis, Princeton University, 2008.

[9] M. Wright and A. Freed. Open sound control: A new protocol for communicating with sound synthesizers. In *Proceedings of the 1997 International Computer Music Conference*, pages 101–104. International Computer Music Association San Francisco, 1997.

[10] https://github.com/markcerqueira/soundCraft