The Eighth Plague *for electronic jar*

Tim O'Brien
Stanford CCRMA MST
119 Quillen Ct. #119
Stanford, CA 94305
tsob@ccrma.stanford.edu

Ilias Karim
Stanford CCRMA MST
Stanford, CA 94309
ilias.karim@gmail.com

Helen Chavez
Stanford CS-HCI UG
Stanford, CA 94309

ABSTRACT

We present The Eighth Plague, an embedded Linux device and related software framework which achieve a simulacrum of human/computer improvisation. The metaphor we chose is the "ghost in the machine," which invokes by analogy the parallelism between mental activity and physical action. We employ a swarm simulation algorithm as an approximate model of human mental activity in an improvisatory context.

The key concept is that the user's interpretations of the interactions are speculative. Manipulation of the device influences the output nonlinearly, and the mechanisms by which this influence is exerted are, at least at first, unclear. This produces a trajectory of user interaction, which starts with mystery, progresses to discovery, and finally arrives at collaboration and improvisation with the device. The result is a unique audio-visual dialogue with a simple, inanimate object.

In this paper we report the details of how we created this device and software environment.

Keywords

Ghost in the Machine, Pure Data, SuperCollider, Python, Swarms

INTRODUCTION

We were captivated with the idea of the "ghost in the machine," some latent personality and intentionality which seems to crop up in the systems we build. The ghost in the machine represents our willingness to imbue a mechanical or digital device, perhaps one of our own making, with an emergent motivation and behavior when it deviates from the designer's intentions. Perhaps the most extreme and archetypal example of this phenomenon is HAL, the computer system in Stanley Kubrick's *2001: A Space Odyssey* (as well as in Arthur C. Clarke's *Space Odyssey* series), which ends up turning on its human creators.

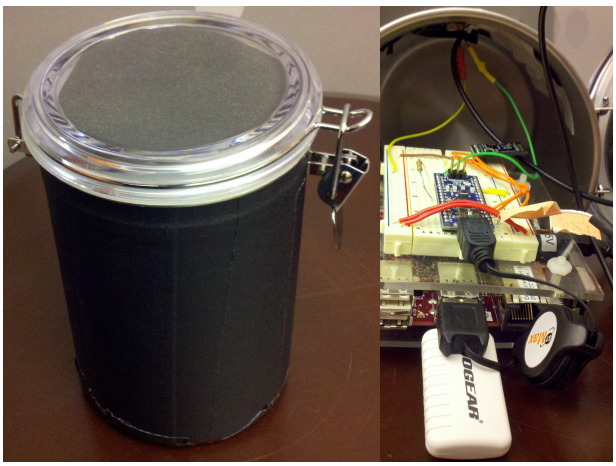
We chose to simulate the ghost in the machine overtly. While we attempted to preserve a sense of foreboding unpredictability (*a la* HAL), we were inspired by toys such as Mattel's Furby. The Furby simulates pet/friend interactions so deftly that children feel discomfort when causing it to express negative emotions (such as turning it upside down) [1][2]. Our aim, similarly, was to create the impression of a responsive, living entity with which to interact. The product of such improvisatory interaction is a musical and visual piece suited for performance, of which the thoughts and emotions of the human performer, on display as (s)he negotiates a dialogue with the device, are an integral part.

In the following sections, we describe the hardware and software components of this system.

DEVICE

Design

A simple jar that could be found in your kitchen was taken and transformed into a black jar filled with electronics. The jar is designed to be small enough to hold in one hand and light enough to move gesturally with minimal effort.



Fg.1: The Eighth Plague jar

Construction

Black tape is used to cover the metal, which is more striking visually and is easier to grip with one hand. Black foam cushion is used inside the lid to hide the contents of the jar. Inside the jar is a BeagleBoard-xM, an Ardiuno, a USB battery, a USB Wi-Fi adapter, and sensors. Sensor data is fed through the Arduino to a Pd patch running on the BeagleBoard. The Pd patch then transmits OSC messages through the USB Wi-Fi adapter through a local network.

The BeagleBoard and Arduino/breadboard are mounted on two pieces of acrylic. Cork board is used to stabilize both this assembly and the battery.

Sensors

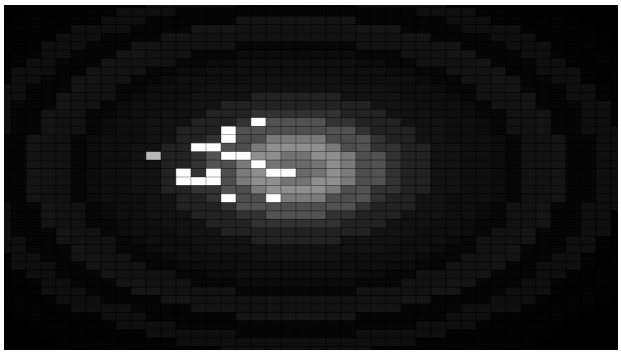
An accelerometer, a piezo disk, and a vibration motor are used to detect the jar's position and motion; physical impulses to the jar's surface; and to provide haptic feedback, respectively.

CLIENT SOFTWARE

The device's client software runs on the BeagleBoard-xM with a mounted Arduino, wireless USB dongle, and USB power supply. The client software is comprised of a single Pure Data patch, which is automatically started when the device boots.

Pure Data

The Pure Data patch is responsible for handling communication with the Arduino, polling and processing the Arduino's sensor data, and communicating with the server. It detects strong jerks of the accelerometer and buzzes the haptic motor in response. These jerk and polled accelerometer data are sent over the Wi-Fi network in OSC messages to the software running on the server.



Fg.2: Screenshot of swarm visualization

SERVER SOFTWARE

Python

We implement the swarm simulation in Python. The simulation is a based upon Craig Reynolds's original Boids algorithm [3] with many of the modifications described in [4]. Specifically, the swarm is multidimensional to an arbitrary degree. This is useful for us because the dimensions in swarm space are interpreted in our sound generation (SuperCollider) module as corresponding to musical parameters such as pitch, loudness, duration, panning, inter-onset-interval, etc.

Moreover, we adopt the dimensional decoupling described in [4]. That is, swarm behavior (the position, velocity, acceleration of each swarm element, or Boid) is calculated independently in each dimension. This is appropriate in a musical context, since we do not need (or want) the frequency, for example, to affect the loudness.

Our Python simulation is agnostic of the musical connotations of each dimension except one: the inter-onset interval. The swarm simulation is note based, and each note is generated when the Python swarm sends an OSC message containing the swarm's centroid values for each dimension to SuperCollider for sonification. The timing of the first note is randomly set, at which time the swarm's centroid value along the last dimension (corresponding to inter-onset interval) determines the time of the next note.

Interactivity with the swarm is achieved by implementing an attractor element in each dimension of swarm space, as in [4]. Each Boid is subject to an acceleration toward this element, in addition to the other Boid forces of attraction to the swarm centroid and collision avoidance. The attractor position for each dimension is set via periodic OSC messages mapping the sensor input to swarm space.

SuperCollider

Our SuperCollider component instantiates several synths and sets up a framework for receiving and sonifying OSC note input from the Python swarm.

Additionally, SuperCollider plays a distinctive sound and activates a different combination of synths each time it receives a "jerk" message from the Eighth Plague device. In performance, we found it useful to designate a set of modes (i.e. active synths and particular scale/interval mappings) over which the user may cycle by jerking the device. A previous version of the SuperCollider module used random activations of the synths on each jerk, which tended toward more dissonant results.

Processing

We use Processing to visualize two different aspects of the system: 1) the state of the swarm and 2) the audio output. The position of each Boid is sent to Processing with OSC messages. These positions are shown on top of an FFT visualization of the audio output. The visualization cycles through different colors, radius resolutions, and grid shape whenever the jar is jerked.

ACKNOWLEDGMENTS

Special thanks to Edgar Berdahl and Wendy Ju for their instruction this fall, and their work on Satellite CCRMA. Spencer Salazar provided additional support and guidance, and the greater CCRMA community was incredibly supportive.

REFERENCES

[1] Furbidden Knowledge. *Radiolab,* s. 10, ep. 1. http://www.radiolab.org/2011/may/31/furbidden-knowledge/ accessed December 8, 2012.

[2] Turkle, S. *Alone Together: Why We Expect More from Technology and Less from Each Other*. Basic Books, 2011 .

[3] Reynolds, C. (1987). "Flocks, herds and schools: A distributed behavioral model.", *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (Association for Computing Machinery): 25–34, doi:10.1145/37401.37406, ISBN 0-89791-227-6

[4] Blackwell, T., Young, M. (2004). Self-organised music. *Organised Sound*, 9(02). doi:10.1017/S1355771804000214

Appendix

Source code may be found online at https://github.com/iliaskarim/EighthPlague