Coding Sound, Crafting Circuits: Synthesizer Design as Critical Making[1]

1. The Sounds in the Machine

Early arcade video games were works of engineering more than programming, and their soundtracks were no exception. In an oft-repeated anecdote, game designer Al Alcorn describes the pragmatics of crafting a very simple repertoire of sounds for his pioneering arcade game *Pong* (1972):

> People have talked about the sound, and I've seen articles written about how intelligently the sound was done and how appropriate the sound was. The truth is, I was running out of parts on the board. Nolan [Bushnell, the founder of Atari] wanted the roar of a crowd of thousands—the approving roar of cheering people when you made a point. Ted Dabney told me to make a boo and a hiss when you lost a point, because for every winner there's a loser. I said "screw it, I don't know how to make any of those sounds. I don't have enough parts anyhow." Since I had the wire wrapped on the scope, I poked around in the sync generator to find an appropriate frequency or a tone. So those sounds were done in half a day. They were the sounds that were already in the machine (quoted in Collins 2008, 8–9).

Early video games were created by small teams of engineers, often on tight budgets and within short timeframes. And as the anecdote from Alcorn illustrates, their soundtracks were often improvisational, emerging through a process of experimentation and discovery as designers searched for novel means to produce meaningful sounds with limited hardware and computing resources. Rather than dedicated composers or sound designers, the task of creating music and

[1] I am grateful to Marlon Kuzmick and Rob Hart at Harvard University, and R.C. Miessler and Eric Remy at Gettysburg College for their many contributions to my thinking about making, the Arduino platform, the digital humanities, and this unit in particular.

sound effects often fell to an engineer. As William Gibbons (2009, 40-42) has noted, this is one reason why early games so often used existing classical music: because it was widely known, existed in notation (which could be easily programmed in by a non-musician), and was in the public domain.

In these older games, there is little functional distinction between a game's source code and its soundtrack. Well into the 1980s, the code responsible for synthesizing sound or music sat right alongside the code that generated the gameplay or the visual assets. And all aspects of a game's design were dependent upon the affordances and limitations of the hardware for which it was made (see Monfort & Bogost 2009, Camper 2012, and Altice 2015). With limited memory and only a few channels of sound, game soundtracks of the 1970s through the 1990s offer insights into the mechanics of game design and development in a restricted environment. In *Space Invaders* (1977), for example, the minimal but menacing descending tetrachord that constitutes the game's "soundtrack" is tied to the same internal clock that sets the tempo for the game itself. In other words, a game-based variable drives the soundtrack as well (Lerner 2013, 328–330). And closely examining games from the well-documented Nintendo Entertainment System (such as *Super Mario Bros.* or the *The Legend of Zelda*) reveals numerous choices and sacrifices made by designers in order to fit polyphonic soundtracks and a variety of soundtracks into only a few synthesizer channels (Altice 2015, 249–288). Even as sound design has grown more sophisticated, tending towards higher fidelity but less interactive forms of music (such as recorded orchestral soundtracks), some games still rely on sophisticated compositional and programming techniques to ensure that their soundtracks react smoothly to the player's actions, accurately reflecting the state of the game (Medina-Gray 2019).

Studying early video game sound and music thus offers a number of pedagogical opportunities for undergraduate music students. First of all, the conditions of very early game development can be reasonably simulated with simple hardware and software. And the relatively limited programmable sound generators (PSGs) found in consoles of the 1980s and 90s are well-understood and exhaustively documented by both enthusiasts and scholars (See Collins 2008, 12–26). Perhaps the most famous of these is the Nintendo Entertainment System (1985), which was capable of three channels of melodic synthesis, one channel of white noise (often used for sound effects or percussion), and a low-resolution sampler (Altice 2015, 249–288). Given this very limited palette for multi-track sound, designers often had to make decisions about which musical voices would be omitted when a sound effect needed to be played. Students thus have the opportunity to observe how musical textures work in a controlled setting, and to hone their skills in close, deductive listening.

But students need not stop at listening and analysis; given that fostering student creativity within a constrained setting is an important goal of instruction in music theory and composition, working with a simulacrum of early game sound technology offers great potential for helping students to understand basic acoustics and synthesis, the rudiments of music theory, and the possibilities of creative coding and critical making. This essay introduces instructors and students to the sonic capabilities of the Arduino platform (www.arduino.cc), an affordable and open source circuit board compatible with numerous components, sensors, and output devices. Arduino is a modern platform, designed for educational settings, that allows students and instructors to explore--among other possibilities--the same potentialities and limitations that were available to early video game sound designers. The Arduino's affordances allow instructors and students to find connections between music theory, computer science, media studies, and the

history of video game music, making possible a kind of speculative material history (Pinch 2008, Whitson 2015) of game sound design, as students tackle the challenging question of how to wring meaningful sounds from a few centimeters of silicon and some wires—precisely the same question facing Alcorn and his contemporaries (or, for that matter, the same question facing the pioneers of commercial synthesizers in the previous decade (see Pinch 2008, 471–473).

In the context of a general education course on music in video games, the Arduino's sound-producing capabilities allow for hands-on exploration of critical topics, and they offer an avenue for students to exercise their own creativity in order to demonstrate their understanding. Furthermore, by introducing students from all disciplinary backgrounds to a simple means for computational musical study, the Arduino can make theoretical pursuits accessible to those without a significant background in notation, open avenues for interdisciplinary thinking, and lay the groundwork for further study in music informatics. Combining academic study with a hands-on, physical intervention, this unit draws inspiration from what is often called the "maker movement," and engages with Matt Ratto's (2011) concept of *critical making*. In this article, I will describe the context of the course, outline a sample unit on designing synthesizers with Arduino, and argue that academic courses in music--for both majors and general education students--have much to gain from engaging with current discourses on critical making and the digital humanities.

2. Technology as Music Appreciation in a Course on Video Game Music

I often teach a general education course entitled "Music in Video Games: Style, Technology, and Culture." Like many music appreciation classes, the course faces the challenge of introducing basic musical concepts alongside an extensive catalog of stylistically diverse repertoire, and

helping students to understand a broad cultural context for the material under consideration. In my course, the period being studied is only half a century instead of half a millennium or more, but its stylistic and technological paradigm shifts sometimes feel as epoch-making as the emergence of polyphony, opera, or sound recording. Little more than a decade elapsed between Al Alcorn squeezing a few waveforms out of a buzzer for *Pong* (1972) and the famous five-track soundscapes of the Nintendo Entertainment System (released 1983; see Altice 2015, 262–264); another decade would find symphonic music being played back in CD-quality audio on home computers and consoles. The history of computing and sound technology is thus an essential context for a class on game music.

In order to bring some rudiments of music theory into contact with that history, I have my students explore basic sound synthesis, acoustics, and circuit design through the Arduino platform. Not only does this approach offer students a hands-on simulation of the kinds of problems that were faced by the sound programmers of early video games, but it also offers an alternate way for students to interact with sound: while I introduce staff notation in the early sessions of the class, most of them have not worked with it before. Understanding how pitch and rhythm can be numerically described outside of notation gives these students two avenues for understanding the same content. This experience benefits the experienced musicians and music majors who sometimes take the class as well: it offers them a new challenge as we cover material that would otherwise be rudimentary for them. Finally, a unit based around coding takes advantage of the fact that many college students have had some exposure to programming, whether it was in high school (where some have encountered the Arduino before), other general education classes, or in their majors. This connection helps to strategically situate the class

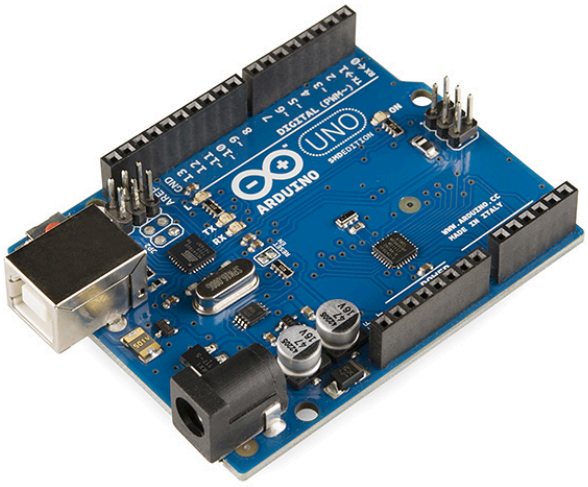
within an increasingly interdisciplinary and STEM-oriented environment within colleges and universities.

3. Making Sound with Arduino

Arduino is an open-source hardware and software platform designed to allow students, enthusiasts, and professionals to build simple digital devices. Projects built with Arduino combine computer code (written in a modified version of the C++ programming language) with hands-on circuit design. Able to interface with any computer via USB, and consisting of a simple circuit board with a variety of ports (usually called "pins") for input and output, Arduino boards can send and receive data to and from a variety of sensors and devices (see Figure 1 and Video Example 1). Common input devices include buttons, dials, light sensors, gyroscopes, and infrared sensors; common outputs include lights, buzzers, speakers, and small digital screens. Arduinos are often sold in "kits" that combine a circuit board with a selection of wires, sensors, lights, and other components. The platform is widely used in educational settings, and offers both students and professionals a broad array of creative possibilities. The company provides an extensive set of tutorials and reference documents, and an active worldwide enthusiast community circulates tips and examples freely.

Video Example 1: A Brief Introduction to the Arduino's Hardware [Embed: https://youtu.be/0YFrmC6X_ZM]

Figure 1: An Arduino Uno board (Image courtesy of Spark Fun, via Wikipedia)



When introducing my students to the Arduino platform, I make use of several of those available examples. After a brief introduction to the platform, we work through building a simple device such as a light switch. We then explore the device's sound production capabilities by introducing two of the Arduino's built-in sound demonstrations: Tone Keyboard and Tone Melody.[2] Finally, I demonstrate a few simple devices I have designed, and show them how each is based on simple extensions of the basic principles they have learned.

The "Tone" command is central to these activities. Through it, the Arduino becomes a tool for both pedagogical demonstration, and creative exploration. Tone asks the Arduino to output a simple square wave at a given frequency. For instance, the command:

```
Tone (8, 440)
```

[2] The full directory of Arduino's sample programs can be found at https://www.arduino.cc/en/Tutorial/BuiltInExamples. A playlist of my own instructional videos from this class can be found at https://www.youtube.com/playlist?list=PL3UxIMtbmt8UqOJHo8betFfLjE0Aa4Frk.

sends a signal to a speaker or buzzer attached to pin #8, causing it to create a tone at 440Hz—the note A, in common Western tuning conventions. Appropriately, the buzzers available to be used with an Arduino are extremely similar to the ones used in early arcade cabinet construction; Figure 2a shows a schematic of a piezoelectric buzzer from KC Collins's (2012) study of early arcade sound, and Figure 2b shows the piezo buzzers that can be used by the Arduino.

Figure 2a: Schematic of a piezoelectric buzzer from an early arcade game (Collins 2012, 123)

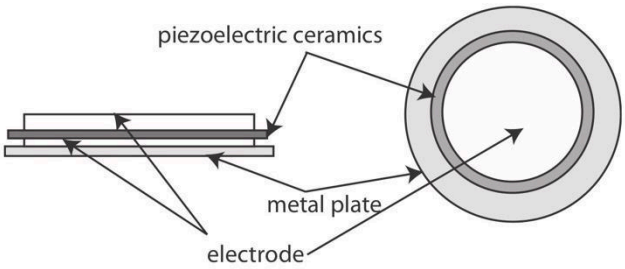


Figure 2b: two piezo buzzers from an Arduino kit

The Tone command also comes with a rudimentary rhythmic syntax: a third term within the command allows a programmer to specify the duration of the tone in milliseconds.

```
Tone (8, 440, 500)
```

would generate the same frequency via the same buzzer as before, but now the tone would last for 500ms, or half a second. Through this additional parameter, the Tone command can be used to program a melody, and exploring the ways in which Arduino's developers have encoded melodies is an extremely useful exercise that gives students the opportunity to interrogate how notation and music theoretical concepts offer us access to a highly detailed sound world.

As noted, the Tone command works by specifying the pitch of the tone in hertz (vibrations per second) and the duration in milliseconds. This numerical precision can be useful—it is easy to demonstrate how the interval of an octave is a 2:1 ratio, for instance. By building their own simple synthesizers and experimenting with simple acoustic ratios, students can hear first-hand how intervals and intonation work. Beyond the first introductory session, however, it can be tedious to enter the frequencies for each note of a demonstration. (And while it is useful for my students to understand the basics of acoustics, it is not necessary for them to grapple with temperament—an issue that would quickly arise in all but the simplest of melodies).[3] The Arduino's solution to this is a library (an additional file that can be called upon by a program) called "pitches.h," which is incorporated into most sound demonstrations. Pitches.h is a list that turns a selection of frequencies into variables written as note names and octaves in scientific pitch notation. One of the Arduino's sample programs, created by Tom Igoe (2011), demonstrates how pitches.h can be used to encode the "shave and a haircut" jingle:

[3] Ironically, intonation does have a role to play in the history of video game music: the Atari 2600 console used an idiosyncratic division of a fundamental frequency to generate its famously incomplete gamut. See Collins (2008, 21-24).

```
#include "pitches.h"

// notes in the melody:
int melody[] = {

NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3, 0, NOTE_B3,
NOTE_C4
};
```

This code excerpt encodes the melody as an "array," a data structure that can then be called as a complete statement. This array, however, includes only the melodic notes; it does not give them rhythmic values.[4] To add those, Igoe uses two additional commands. The example code first assigns a standard note duration to each pitch in the above series, using 4 to indicate a quarter note and 8 to indicate an eighth note (the line that begins with two slashes is a comment within the code, explaining how the variable works):

```
// note durations: 4 = quarter note, 8 = eighth note, etc.:
int noteDurations[] = {

4, 8, 8, 4, 4, 4, 4, 4
};
[...]
int noteDuration = 1000 / noteDurations[thisNote];
```

[4] The digit 0, in the middle of the melody, functions as a short rest.

Then, he adds a line that performs each of these durations as a fraction of a full second (e.g. 1000ms/8 yields an eighth note that is 125ms long, while 1000/4 yields a quarter note that is 250ms long). As shown in Video Example 2, executing the "Tone" command on these arrays will cause the Arduino's buzzer to perform the jingle in rhythm, and studying the code involved offers students insight into the numerical foundations of rhythmic notation. Furthermore, the instructor or the students are able to manipulate the values in the code, and students can hear for themselves what happens when a single pitch is altered or a rhythmic value is halved or doubled, or even randomized.

Video Example 2: Creating "Tone Melody" [embed: https://youtu.be/CtlmIFWbxGY]

Departing from there, I ask students to chronicle their experiments with the Arduino through a two-part project. For the first half of the assignment, students must construct and execute one of the sample projects that comes with their Arduino set, and document what happens when they change one or more of the variables. Then, once they have become comfortable with the basics of scripting and manipulating hardware (aided by reference materials, tutorial videos, and in-class workshops), I ask the students to create their own original sound-producing device.

In the second half of the project, students are encouraged to explore the creative possibilities that arise when the Arduino's Tone command is combined with the device's other capabilities. The Tone can be triggered by an input, for example, so that whenever a button is pressed or a sensor is triggered, the Arduino makes a sound. "Tone" will also accept a variable in place of a numerical frequency. Most Arduino sensors operate by returning numerical values

between 0 and 1023—a span that, if converted to frequency, is mostly congruent with the frequency span covered by voices and common acoustic instruments (a little more than four octaves, roughly equivalent to the span of a 49-key MIDI keyboard). This congruence makes it possible to design simple, experimental synthesizers based on nearly any input. Students can build a device, for example, that raises or lowers pitch as they turn a dial (I often share my own experiments with this capability as a further example; see Video Example 3). Or, using a proximity sensor, they can build a rudimentary Theremin, whose pitch is controlled by how close one's hand is to the sensor. In the context of a course on the history of video game music, these hybrid explorations in circuit design, coding, and composition mirror suggestively the kinds of experiments undertaken by early game sound designers, like Alcorn with his board full of transistors. In that spirit, the best student projects that I have seen come from this unit combine two sensors or controls in unexpected ways, or use code creatively to find ways to record, manipulate, and produce music as data.

Video Example 3: The "Siren Synth," a project that uses a dial to control pitch. [Embed: https://youtu.be/dvRDdM2fQac]

For more advanced students, this introduction to computational thinking about music prepares them to explore programs like Opus Modus, Music21, or the Humdrum Toolkit. Or using the Arduino, they might be ready to have discussions that bring music and code together to address the many questions raised by the simple "out of the box" examples. For instance, "Tone Melody's" reliance on fractions over 1000ms has the effect of setting the tempo at 60 beats per minute. But what would be necessary to get the example to run at 90bpm, or 74, or 110? How

might the variable(s) be manipulated further? Advanced students—even up to the graduate level—might benefit from using the Arduino to work with various unequal temperaments. An instructor might ask their students to encode a melody in raw frequencies and attempt to solve intonation problems as they arise, or to create their own version of pitches.h that represents some form of meantone temperament or just intonation.

4. Reflections: Critical Making as a Model

Using the Arduino to freely explore the musical potential of circuit design and exploratory coding—whether in pursuit of a speculative material history of early game sound design, or as a twenty-first-century instrument of music-theoretical exploration (see Rehding 2016, O'Hara 2021)—the experiments described here can be thought of as a branch of *critical making*. The concept of critical making, as described by Matt Ratto, embodies "a desire to theoretically and pragmatically connect two modes of engagement with the world that are often held separate—critical thinking, typically understood as conceptually and linguistically based, and physical 'making,' goal-based material work" (2011, 253). Ratto offers a three-stage model for critical making. It begins with a survey of academic literature and an initial compilation of significant ideas or concepts; continues with a collaborative process of designing and building prototypes; and leads to "an iterative process of reconfiguration and conversation, and reflection" (ibid).

Critical making often finds its (inter)disciplinary home in the digital humanities, spread out between academic departments, libraries, instructional designers, and the hybrid collaborative spaces (often called "makerspaces" or some form of "lab") now found on many campuses (Mersand 2020, Mestre 2020, Wong & Partridge 2016). When applied to music, the

concept is simultaneously a radical challenge to music pedagogy, and a simple continuation of business as usual. On one hand, there is rich resonance between Ratto's critical making and the practices that are already widespread in schools of music. Music schools excel at practical pursuits, from the private lessons, small group coachings, and collaborative rehearsals that are the bread and butter of music performance programs; to composition studio classes and workshops; to interdisciplinary theatrical, artistic, or digital endeavors. On the other hand, the relationship between academic pursuits (i.e. music theory and musicology classes) and applied study or ensemble performance is often a source of significant curricular and logistical tension. We could easily map that gulf onto Ratto's dichotomy between conceptual/critical thinking and physical "goal-based material work," though we ought to resist partitioning our activities so neatly. Indeed, the resistance to presumed disciplinary divisions between theory and practice, or between reflection and creation, offered by an ethos of critical making ought to guide our activities as scholars, teachers, and performers; embracing the notion of critical making in musical study is one way to articulate the ideal of the "educated," "thinking," or "well-rounded" musician commonly spoken of in departmental mission statements and curricular documents. And it should be noted that there are many examples of institutional structures that take seriously the integration of theory and practice, such as music theory curricula that combine written theory, model composition, ear training, and keyboard skills within a single class.

In other ways, however, music scholars have much to learn from embracing the concept of critical making, particularly with regard to introductory classes. We do not always expect students in introductory, general education classes to make music, and if we do, we frequently draw upon skills they already had before enrolling in the course. Working with simple synthesizers allows students to build, work with, and eventually design their own devices from

scratch, drawing upon and extending their knowledge of musical fundamentals and the affordances of code. Doing so encourages students of all levels to adopt an attitude of exploratory, intellectual play, probing received concepts and remixing them into new forms.

Using the Arduino to make sound also speaks directly to the ways in which musical sound is *theorized*, and it gives students from any academic background the opportunity to understand and interrogate how notation and music theoretical concepts offer us access to a highly detailed sound world. For instance, the unit underscores the fact that both rhythmic proportions and consonant intervals revolve around simple ratios, and encourages students to play with the implications of this fact. As Jeremy Day-O'Connell (2019) argues, the academic study of music theory (even at the undergraduate level) ought to include careful attention to how our labelling systems make sound legible, and how our understanding of musical objects influences the way we study, understand, perceive, and perform them. Day-O'Connell demonstrates this outlook through several examples, including one that contrasts a major sixth with a diminished seventh in different tonal contexts by asking students to sing the diatonic steps that comprise the intervals. The lesson, he writes, is to show how the concept of an interval has greater explanatory power than simply counting semitones. As David Lewin (1968, 61) argues, music theory "attempts to describe the ways in which, given a certain body of literature, composers and listeners appear to have accepted sound as conceptually structured, categorically prior to any one specific piece." This is much the same lesson that is taught when students learn how to manipulate time and pitch on the Arudino: each parameter can be accessed as pure data (in the form of frequency in hertz or duration in milliseconds), but is much easier to understand and use when translated into a more human-readable format (such as the note names from pitches.h). While both raw forms of musical data can be useful, students come to understand how

adding a layer of interpretation—theory, in other words—on top makes them more legible and easier to use for practical music-making and theorizing.

References

Altice, Nathan. 2015. *I Am Error: The Nintendo Family Computer / Entertainment System Platform*. Cambridge, MA: MIT Press.

Camper, Brett. 2012. "Color-Cycled Space Fumes in the Pixel Particle Shockwave: The Technical Aesthetics of *Defender* and the Williams Arcade Platform, 1980-82." In *Before the Crash: Early Video Game History*, ed. Mark J.P. Wolf. Detroit: Wayne State University Press.

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

_____. 2012. "One-Bit Wonders: Video Game Sound Before the Crash." In *Before the Crash: Early Video Game History*, ed. Mark J.P. Wolf. Detroit: Wayne State University Press.

Day-O'Connell, Jeremy. 2019. "Putting the Theory Back in 'Music Theory.'" *Engaging Students* 7. https://engagingstudentsmusic.org/article/view/7368.

Gibbons, William. 2009. "Blip, Bloop, Bach? Some Uses of Classical Music on the Nintendo Entertainment System." *Music, Sound, and the Moving Image* 2/1: 40–52. https://www.jstor.org/stable/10.5406/musimoviimag.2.1.0040.

Holman, Will. 2015. "Makerspaces: Towards a New Civic Infrastructure." *Places*. http://www.placesjournal.org/article/makerspace-towards-a-new-civic-infrastructure. Accessed 1 June 2021.

Lerner, Neil. 2013. “The Origins of Musical Style in Video Games, from 1977 to 1983.” In *The Oxford Handbook of Film Music Studies*, ed. David Neumeyer. New York: Oxford University Press.

Lewin, David. 1968. “Behind the Beyond: A Response to Edward T. Cone.” *Perspectives of New Music* 7/2: 59–69. https://doi.org/10.2307/832293.

Medina-Gray, Elizabeth. 2019. “Analyzing Modular Smoothness in Video Game Music.” *Music Theory Online* 25/3. https://www.mtosmt.org/issues/mto.19.25.3/mto.19.25.3.medina.gray.html.

Mersand, Shannon. 2020. “The State of Makerspace Research: a Review of the Literature.” *TechTrends* 65: 174–186. https://doi.org/10.1007/s11528-020-00566-5.

Mestre, Juliana. 2020. “The Troubling Trend of Academic Makerspaces.” *Public Services Quarterly* 16/4: 273–279. https://doi.org/10.1080/15228959.2020.1819937.

Monfort, Nick, and Ian Bogost. 2009. *Racing the Beam: The Atari Video Computer System*. Cambridge, MA: MIT Press.

O’Hara, William. 2021. “Digital Ecologies of Twenty-First Century Music-Theoretical Instruments.” *Everywhere It is Machines* Lecture Series, Royal Holloway, University of London. May 6, 2021. https://www.youtube.com/watch?v=GeLnGlmznFY.

Pinch, Trevo. 2008. “Technology and Institutions: Living in a Material World.” *Theory and Society* 37: 461–483. https://doi.org/10.1007/s11186-008-9069-x.

Ratto, Matt. 2011. “Critical Making: Conceptual and Material Studies in Technology and Social Life.” *The Information Society* 27: 252–260. https://doi.org/10.1080/01972243.2011.583819.

Rehding, Alexander. 2016. “Instruments of Music Theory.” *Music Theory Online* 22/4. https://mtosmt.org/issues/mto.16.22.4/mto.16.22.4.rehding.html.

Whitson, Roger. 2015. “Critical Making in the Digital Humanities.” In *Introducing Criticism in the Twenty-First Century*, ed. Julian Wolfreys. Edinburgh: Edinburgh University Press.

Wang, Anne, and Helen Partridge. 2016. “Making as Learning: Makerspaces in Universities.” *Australian Academic & Research Libraries* 47/3: 143–159. https://doi.org/10.1080/00048623.2016.1228163.