The Sincerest Form of Flattery: Large-Scale Analysis of Code Re-Use in Atari 2600 Games

John Aycock
Shankar Ganesh
aycock@ucalgary.ca
sankarasubramanian.g@ucalgary.ca
University of Calgary
Calgary, Alberta, Canada

Katie Biittner
MacEwan University
Edmonton, Alberta, Canada
biittnerk@macewan.ca

Paul Allen Newell
Independent researcher
United States

Carl Therrien
Université de Montréal
Montréal, Québec, Canada
carl.therrien@umontreal.ca

ABSTRACT

The Atari 2600 was a prominent early video game console that had broad cultural impact, and possessed an extensive catalog of games that undoubtedly helped shape the fledgling game industry. How were these games created? We examine one development practice, code re-use, across a large-scale corpus of 1,984 ROM images using an analysis system we have developed. Our system allows us to study code re-use at whole-corpus granularity in addition to finer-grained views of individual developers and companies. We combine this corpus analysis with a case study: one of the co-authors was a third-party developer for Atari 2600 games in the early 1980s, providing insight into why code re-use could occur through both oral history and artifacts preserved for over forty years. Finally, we frame our results about this development practice with an interdisciplinary, bigger-picture archaeological view of humans and technology.

CCS CONCEPTS

• **Applied computing** → **Computer games**; **Archaeology**; • **Social and professional topics** → **History of software**; • **Software and its engineering** → *Software reverse engineering*.

KEYWORDS

Atari 2600, game development, binary reverse engineering, archaeogaming, empirical study

ACM Reference Format:
John Aycock, Shankar Ganesh, Katie Biittner, Paul Allen Newell, and Carl Therrien. 2022. The Sincerest Form of Flattery: Large-Scale Analysis of Code Re-Use in Atari 2600 Games. In *FDG '22: Proceedings of the 17th International Conference on the Foundations of Digital Games (FDG '22), September 5–8, 2022, Athens, Greece.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3555858.3555948

FDG '22, September 5–8, 2022, Athens, Greece

ACM ISBN 978-1-4503-9795-7/22/09…$15.00
https://doi.org/10.1145/3555858.3555948

1 INTRODUCTION

A book by the Reverend C. C. Colton published in 1837 was replete with aphorisms, one of which was 'imitation is the sincerest of flattery' [14, p. 113]. From the point of view of video games, Colton was prophetic, presaging the early video game industry where highly derivative games – and outright knockoff games – were common.[1] Judgments about the similarity between two games is typically based on the games' look and feel, which is arguably a superficial view. What about similarity underneath the hood, in the game code itself?

Specifically, we focus here on *code re-use*, where code in one game can be found in another game. Code re-use is commonplace now, with web sites like Github and Stack Overflow, but back in the 1970s and 1980s, information flowed much differently than it does today. Game programmers of that era might have technical books or manuals, possibly fellow programmers or a user group to exchange tips with, maybe even have a peek at other programmers' code by reverse engineering it. And both then and now, programmers frequently re-use their own code from project to project. By examining game code, there should be evidence of this development practice.

Our work centers on game code for the the Atari 2600, originally called the Atari VCS. The Atari 2600 was an early game console released in 1977 that had a surprisingly long lifespan: the 2600 did not cease production in its various incarnations until 1992, long after it had been surpassed by other devices. As for why we are studying *this* console and *this* corpus of games, however, it is necessary to delve into the context of the the console and the company that produced it.

Of all the big players in early videogame history, Atari has attracted the most historical consideration. Game historians and fans are unable to keep this enamored corporation buried in the ground – quite literally, as made evident by the 2014 exhuming in Alamogordo, New Mexico. A fraction of the hundreds of thousands of unsold cartridges, downgraded to the status of trash by their very

[1] Thank goodness derivative and knockoff games are a thing of the past in 2022.

own makers and buried in 1983, were recovered in a documentary-driven event involving on-site archaeologists [39].

Atari, the company, became the symbol of the extraordinary growth of a new cultural industry based on technological wonders. Visuals from the era, such as Atari box art, hold enough nostalgic fascination to warrant the release of coffee table books [27]. Its history is well documented at this point, with many key figures providing interviews. The Strong National Museum of Play holds an impressive repository on the company, ranging from arcade cabinets to arcade flyers and corporate records. From the standpoint of production studies, Atari is the most expansive playground one could hope for.

Many recent historical accounts make good use of these documentation efforts to revisit some of the most well-known episodes in videogame history. Michael Z. Newman named his history of early videogames the 'Atari Age' [36]. Anne Ladyem McDivitt explores the fascinating history of early pornographic software on consoles and computers in a chapter entitled 'Atari Generation' [32]. Atari is personified through this sort of vocabulary. In return, this Atari "persona" appears to define the various people who were part of the craze, and even pit them against future "generations." For instance, Jamie Lendino sets out to present the 2600 as the great progenitor of the new cultural industry, his stated objective to help protect some sort of clan pride [28]. If, as Newman argues, 'what makes early video games distinct from games in later periods in the history of the medium is precisely this lack of stable identity' [36, p. 2], then one might wonder why we are so quick to elevate Atari as the poster child for videogame's infancy and first major crisis (the infamous 1983 crash), and why Atari acts as both as the child prodigy and toxic fratboy, subsuming all the good and bad happening at the time.

Further, there is a distinct lack of diversity in that we mostly exhume the same titles over and over. *Asteroids*, *Combat*, *Adventure*, *Pitfall!*, *E.T. the Extra-Terrestrial*, *Pac-Man*, *Custer's Revenge*, *Space Invaders* and a few others get the lion's share of the historical imagination, and understandably so: they had significant cultural impact on gaming culture at the time. Newman and McDivitt do provide engaging overviews, but the scope of these works restricts the amount of close readings one can integrate. Within 252 pages, Lendino manages to provide one-pagers (on average) of context and appreciation for over 90 games, an impressive figure, highlighting clever programming and innovations. Still, one might wonder if this figure, less than 15% of known games released on the platform, truly represents 'every significant game' [28, p. 12].

In this paper, we use novel tools for digital inspection to examine not just a small, hand-selected set of games for the Atari 2600, but a corpus containing nearly *all* games for the 2600. A distinguishing feature of this work is its interdisciplinarity, with co-authors representing three different areas of research and scholarship. Computer science facilitates the technical analysis of the 2600's game code; anthropology and archaeology situates the technical work in terms of the relationship between humans and their technology; game history contextualizes this particular game console and its corpus of games into the bigger picture of games as cultural artifacts. This work also involves a co-author who developed Atari 2600 games in the early 1980s, and the development artifacts he preserved along with his oral history contributions give us insight into development practices that could involve code re-use. Note that any oral history is clearly labeled with his initials to separate subjective from objective.

After the related work in the next section, Section 3 presents the technical aspects of the work and its findings. This is followed by the case study in Section 4 and an anthropological-archaeological discussion in Section 5, prior to our conclusions.

2 RELATED WORK

There are many examples where a single game's implementation is studied in depth, either starting with source code (e.g., [45]) or reverse-engineering binary code (e.g., [4, 56]), and in fact there may be more activity in the study of single games from enthusiasts than academics. Examining small numbers of games for a specific platform brings in work in the area of platform studies [2, 33, 54]. None of these address the problem of scale, however: there are enormous numbers of games, and limited work that is able to provide insight into all of their implementations. The only work we are aware of that attempts this feat examines a corpus of 132 games produced by an authoring tool for text adventure games [5]. Unlike that work, we do not limit ourselves to games of a single genre, our corpus is predominantly comprised of games that were professionally developed and published, and we are able to study a larger set of games.

There is related work in the realm of software engineering in the guise of code clone detection. In particular, there are code analysis systems that can operate on extremely large code bases [43], although the critical difference is that they operate at the source code level, a luxury we do not enjoy for Atari 2600 games, where having the original assembly language source code is a rarity – we have to work with binary code. And, while there is work on detecting code similarity in binary code [20, 24, 30], that work shares an implicit advantage: it is all relatively recent and is being run on modern binary code. For reference, work proclaiming itself to be 'the first practical clone detection algorithm for binary executables' [42, p. 117] was published in 2009 and analyzed binary code published in the 21st century. Modern binary code is almost exclusively compiler-generated code, and that plus the tendency to separate code and data into different areas in the program are substantial assets; a study of disassembly on modern platforms using non-obfuscated binaries showed that assembly instructions could be identified with exceedingly high if not perfect accuracy [3].

By contrast, the assembly code implementing Atari 2600 games was not compiler-produced, but was hand-written by humans. Due to the constraints of the platform, the code would not be obfuscated, but *would* be highly optimized for both speed and space, and the dividing line between optimized and obfuscated code at that level can be very difficult to discern. Programmers could – and did – mix code and data in the games, and the separation of code and data for analytical purposes has long been known to be an undecidable problem to solve [23]. Therefore any method we use must necessarily be heuristic. Certain optimizations performed by Atari 2600 programmers make the situation worse, in fact: code and data cannot be perfectly separated for analysis where code was being used as data, as in *Yar's Revenge* using bytes of code as

Table 1: Corpus composition by ROM size

Size	Number of ROMs	Percentage
2 KiB	209	10.5%
4 KiB	1141	57.5%
8 KiB	398	20.0%
8448 bytes	76	3.8%
10,495 bytes	3	0.2%
12 KiB	7	0.4%
16 KiB	135	6.8%
16,896 bytes	1	0.0%
25,344 bytes	5	0.3%
32 KiB	6	0.3%
33,792 bytes	2	0.1%
64 KiB	1	0.0%

a surrogate for random data to be displayed onscreen [33] or, less visibly, in Carol Shaw's *River Raid* [48].

More generally, our work can be seen to fit within the scope of *archaeogaming*, a relatively new area of study within the field of archaeology. Reinhard defined archaeogaming as 'the archaeology both in and of digital games' [40, p. 2], and it includes work as diverse as the exploration of narrative space through videogames [53], performing archaeology in virtual game worlds [41], ethics in video game archaeology [17], and the aforementioned dig in New Mexico [39].

3 FINDING CODE RE-USE AT SCALE

The discussion of how we found instances of code re-use in Atari 2600 games is divided into three parts: the corpus composition and code representation, the matching process, and the results.

3.1 Corpus and BAD Code

We gathered three large pre-existing collections of Atari 2600 ROM images from the Internet, which we are using under fair dealing and fair use copyright exemptions for research and commentary. We identified and removed exact duplicates using the images' MD5 checksums[2] which left us with 1,984 distinct ROM images. Table 1 shows the breakdown of the images by ROM size; the corpus is dominated by 2, 4, and 8 KiB images. Most of the oddly-sized images belong to games for the Atari 2600 add-on Starpath Supercharger device that loaded games from cassette and, while they are not ROMs *per se*, they contain code and data that can be analyzed uniformly with the rest.

An important platform constraint for the Atari 2600 was that it could only directly address ROMs 4 KiB or less in size, and games larger than that employed a bank-switching scheme, where hardware in the game cartridge would swap between different ROM chips as directed by software. This impacted disassembly of the code, because "smarter" disassemblers that would attempt to follow control flow of the program fared poorly when they were unable to see the control flow changes caused by bank switching that occurred outside the purview of the CPU.

[2]MD5 is no longer a strong algorithm for security purposes, but it is still sufficient for this de-duplication task.

Instead, the images were all disassembled using a linear-sweep disassembler [13] we built for the 6507 CPU inside the Atari 2600. Only documented 6507 instructions were disassembled, since we reasoned that use of undocumented instructions would have been rare, and this avoided having a number of data values being erroneously interpreted as instructions. Our disassembler looked for sequences of instructions concluding with control transfer instructions like, for example, unconditional branches,[3] and output these instruction sequences in the form of Binary Abstracted Disassembly: "BAD code."

When assembly code is re-used from one game to another, there are some aspects of the code that are likely to stay the same, and some that are likely to change. The exact memory addresses of re-used subroutines or variables would be almost certain to change across games, for instance. BAD code retains the following information:

- 6507 instruction opcodes (and, implicitly, the instructions' addressing mode);
- immediate operands for instructions, which are essentially constant values;
- relative branch offsets;
- fixed memory addresses referring to the memory-mapped custom "TIA" chip in the Atari 2600 responsible for graphics and audio.

Other information, such as non-TIA memory addresses, is discarded. The BAD design is meant to strike a balance between false negatives, where instances of code re-use would be overlooked, and false positives that would be meaningless results. The conservative design choices naturally mean that some code re-use may be missed – for example, an instruction inserted into a loop of otherwise re-used code will likely cause the loop's relative branch offset to change due to the insertion, and consequently its BAD code sequence will not be seen as a match to the original. However, that is an acceptable, and in fact a preferable, result. When two BAD code sequences match, we want to have surety that the code is the same, and not have to make subjective judgments about the intent of any code differences especially when considering a large volume of code.

Figure 1 shows an excerpt of original 6507 code from the game *Pitfall!* and its BAD code equivalent. Each BAD code instruction is normalized into two bytes, regardless of whether the original instruction was shorter (`inx`) or longer (`jsr`). The TIA memory reference (`$1c`) and immediate operand's value (`$06`) are preserved along with instruction opcodes, and the target address of the `jsr` is abstracted away to the placeholder value `$ff`. While most translation from 6507 disassembly into BAD code is straightforward, there is one case subject to postprocessing. If there are two `brk` instructions in a row, the second is interpreted as data rather than an instruction. The reason for this is that two `brk` instructions in a row in legitimate code were highly unlikely, whereas the value 0 (coinciding with the `brk` opcode) appeared regularly in data and threw off the heuristic code/data separation during disassembly.

Finally, we filtered out ROMs that were too similar, using the BAD-code representation of the ROM images to take advantage

[3]The complete set is `jmp`, `rts`, `rti`, and the eight relative branch instructions. The `jsr` jump-to-subroutine instruction was not included as a control transfer instruction because control flow typically returns from the subroutine the `jsr` calls, and `brk` was also excluded for similar reasons.

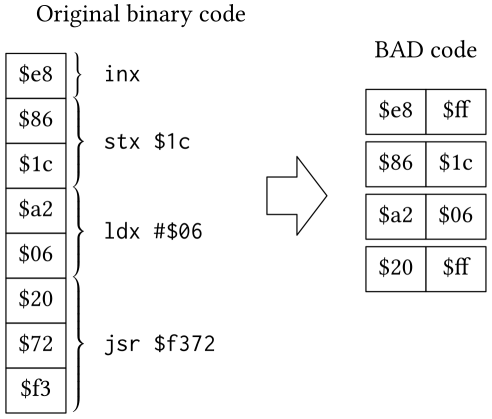


Figure 1: Four instructions from *Pitfall!* and corresponding BAD code ("$" denotes a base 16 number)

of its abstracted form. The original corpus could contain multiple ROMs for a single game: series of ROMs captured during game development; game versions for different television types (e.g., NTSC vs. PAL); re-releases by different publishers. Looking for code re-use in near-duplicate ROMs was unlikely to provide substantial new information, and therefore we wanted to automatically and objectively choose a single exemplar for each different game. Computing the normalized compression distance [29] with zlib compression [18], we rejected ROMs whose similarity threshold fell below 0.5 compared to a ROM already in the corpus. The resulting filtered corpus contained 704 ROM images that we used for the remainder of this work.

3.2 Calibration and Matching

To match BAD code sequences against one another to look for code re-use, we needed to choose a minimum BAD code length N. A value of N that was too short would yield results that would be plentiful, but also uninteresting and hard to argue definitively were true cases of code re-use. As a simple example, the 6507's add instruction always included the processor's carry bit in its computation, and therefore seeing a code sequence that cleared the carry bit followed by an addition was perfectly normal and not code re-use at all. At the other end of the spectrum, selecting a too-large value for N would overlook some code re-use.

A previously found instance of code re-use, identified in an Atari 2600 game through manual analysis thanks to a distinctive bug in the code [6], provided a hard upper bound for N: that instance of code re-use was 21 instructions long. For a lower bound, we reasoned that skilled Atari 2600 programmers would be unlikely to deliberately re-use code within the *same* game unless there was an extremely compelling technical reason to do so, because the amount of ROM space was tightly constrained. We ran two well-known games, *Combat* and *Pitfall!*, through the code matching process against themselves with different sequence lengths to see at what point the number of self-matches declined to 0. Figure 2 shows the results. *Combat*'s BAD code sequence all became unique at $N = 9$, with *Pitfall!* taking longer to drop off but exhibiting a similarly

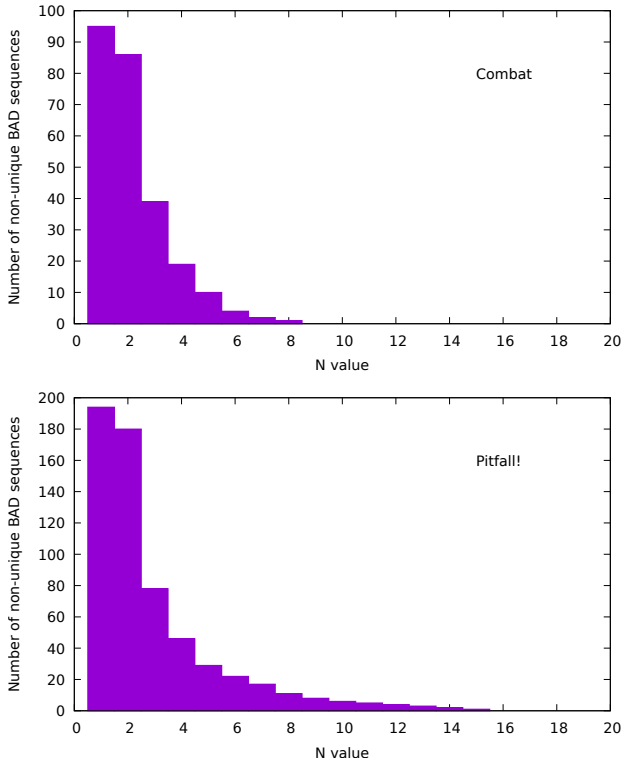


Figure 2: Matching BAD code sequence of varying lengths within the same game, for calibration

shaped curve. Based on this, we chose the value $N = 15$ for the minimum BAD code length.

The matching itself is an $O(n^2/2)$ comparison process between the n ROM images' BAD code. We use suffix arrays [31] to efficiently find BAD code sequences of length N or greater that appear in pairs of ROM images. Where the matches between two ROM images do not overlap, this is uncomplicated, and when overlaps occur, we prefer the longest match. Even with removing overlaps, there are still many matches to consider across the corpus: 26,359. However, many of the sequences recur, and there is a relatively smaller number of 3,221 unique BAD code sequence matches.

It is reasonable to wonder if all these sequences correspond to code, given the limitations of the linear-sweep disassembly that started off the process trying to solve an undecidable problem by distinguishing code from data. We took a random sampling[4] of 10% of the unique, found sequences, and an independent manual analysis was performed on them by two different co-authors. Of those 323 sequences, over 93% were code, a very high accuracy rate. Despite this, we will not be reporting any aggregate code re-use measures in our results below, instead erring on the safe side with results that have been manually vetted.

[4]The randomness was drawn from the `urandom` source on Linux.

3.3 Results

We started by manually analyzing the 25 most frequent unique BAD code sequences in the corpus which, together, made up over 58% of the 26,359 matches in the corpus. The surprising result is that they turned out to be variants of the same two routines, and those routines have very different properties from the code re-use point of view.

The first of the two routines was code to display a sprite that was 48 pixels wide, something which could be used beyond in-game objects for a company logo or a six-digit score. Programming graphics using the Atari 2600's TIA chip was extremely challenging: it required real-time programming, counting the number of machine cycles each 6507 instruction used, in order to get the timing correct. Programmers' code was 'racing the beam' [33], the television's electron beam that was sweeping across the screen to generate the display, and programmers would have to do this for *every* line on the screen. For a 48-pixel-wide sprite, the timing of the 6507 instructions left few options, because the three 6507 registers needed to be completely filled with data to write to the TIA at the appropriate time [25]. It is hard to make a strong argument for code re-use here because, as with the addition on the 6507 mentioned earlier, there were limited ways to accomplish this effect. Wide sprites were not the only situation like this, and one method used to create a multi-object display for the Atari 2600 was even awarded a patent [1].

The second of the two routines, by contrast, is easier to make the code re-use case for, because there were many ways to achieve its goal in 6507 code. The purpose is to calculate two values necessary for horizontal positioning of game objects on the Atari 2600. Game objects' position could not be set directly by writing a numeric value – that would be too easy. Instead, the game code would have to access a TIA location precisely when the television's electron beam was at the spot where the game object should be placed. The tightest loop in 6507 code takes 5 machine cycles per iteration, during which time the electron beam would move 15 pixels on screen; the ability to take the desired horizontal position of a game object and compute its division and remainder by 15 was thus needed for games. The 6507 processor, like many small CPUs of that time, did not have a division instruction, meaning the calculation had to be done in software. Two ways to accomplish this are repeated subtraction and table lookup [25], and a third way is seen in the second frequently used routine. The code sequence is a clever way to compute the division and remainder by 15 without any loops, which we will call `HRCALC`.

The `HRCALC` code appears in David Crane's 1982 game *Pitfall!*, with what appears to be an earlier evolutionary step in Crane's *Canyon Bomber* from 1979. Carol Shaw's game *River Raid* (1982) has a perfect match for the full routine, and the source code for her games is held by the Strong museum [48], meaning that the original source code for `HRCALC` can be seen; this is, in fact, where we take the name `HRCALC` from. While the source code is well documented, unfortunately there is no credit given for the routine, nor would there necessarily have been if it was simply common code used within the company: both *Pitfall!* and *River Raid* were Activision games. To underscore this point, the two games also share some attract-mode code; it and the calculation code are perhaps best thought of as game infrastructure code rather than game code *per se*, and there would be no sense reinventing the wheel for this within the same company. What is intriguing is something we return to in the next section: how did this code get into non-Activision games?

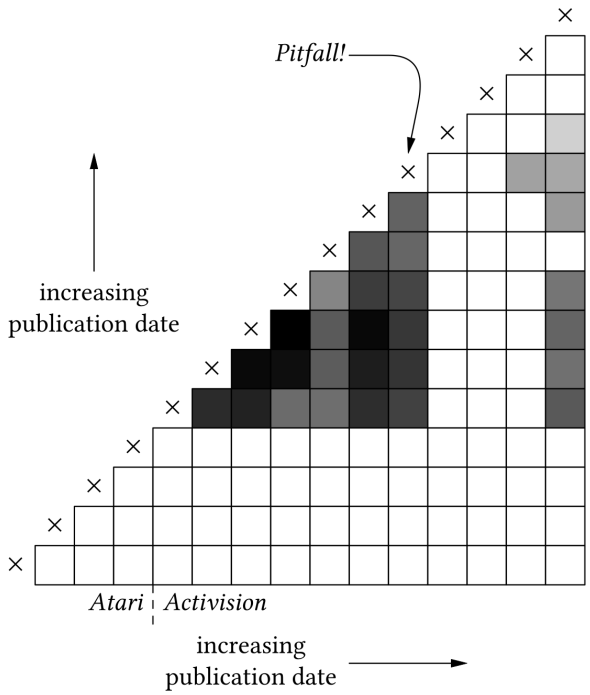


Figure 3: Heat map showing code re-use between David Crane's Atari 2600 games

We can use our system to get a more a targeted view into code re-use practices of both single developers and companies. Figure 3 shows a heat map depicting code re-use between David Crane's games, for instance. We ordered Crane's games, preferring the games' publication date where available; this is admittedly not always precise, and the development date would be better but is both generally unknown and would introduce the additional complicating factor that multiple games could be under development simultaneously. A darker heat map shading (log scale) represents a greater amount of re-used code detected between a game on the X-axis and earlier game releases, with the "×" indicating where a game would be compared with itself. While we are interested in the high-level view, we have marked *Pitfall!*'s location in the plot for reference.

First, it is evident from the heat map that there was code re-use happening within Crane's games. What is perhaps more striking, however, is the fact that the code re-use is not detected between his early games, and starts suddenly four games in, which is telling. Crane started his Atari 2600 development career at Atari, then became one of the co-founders of Activision, where his later game releases occurred. The detected code re-use starts precisely at that corporate boundary: no code re-use between his Atari games, code re-use between his Activision games. There are several possible explanations for this that are not mutually exclusive. Any new platform requires time for a programmer to come up to speed and learn how to best take advantage of it, and the Atari/Activision

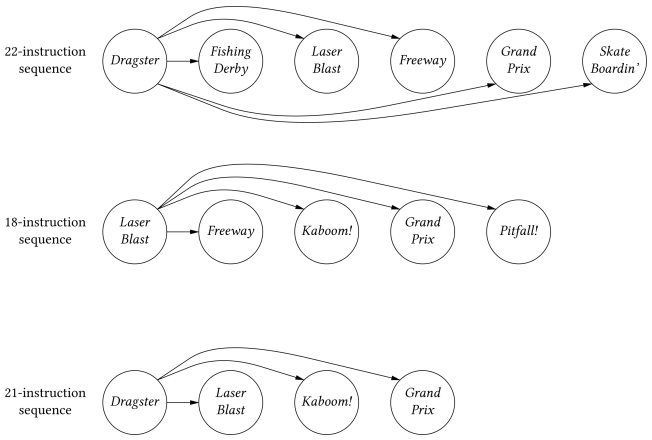


Figure 4: Lineage of three of Crane's most frequently re-used code sequences, with games published earlier on the left

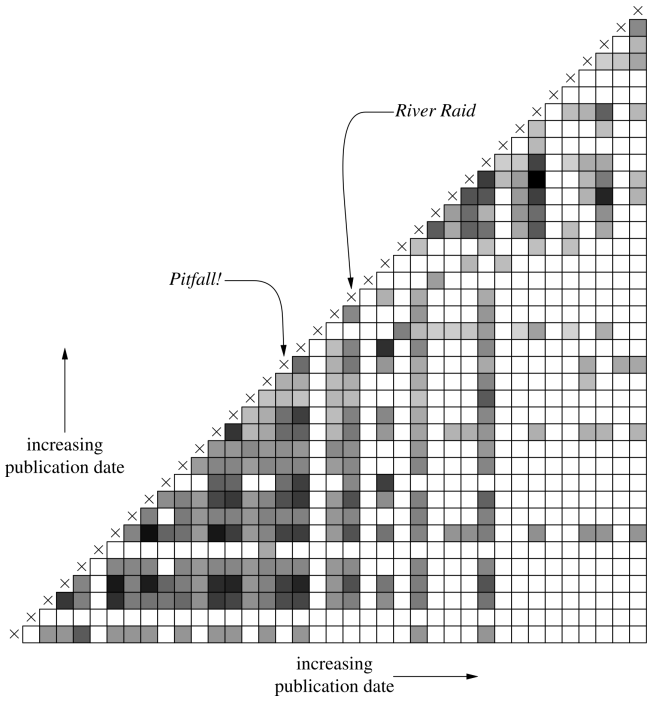


Figure 5: Heat map showing code re-use between Activision's Atari 2600 games

divide may simply be coincidental and followed Crane's learning curve developing games for the 2600 platform. There may also have been a cultural shift in development when moving from Atari, with its established practices and management, to the then-startup environment of Activision. The culture within companies aside, there were likely external influences at play between companies: Crane and the other Atari co-founders of Activision (correctly) anticipated legal trouble from their former employer and ensured that they had a clear separation of intellectual property [52]. One might additionally look for technical reasons, such as the ROM size, but in fact Crane's earliest games at Activision used the same 2 KiB ROMs as his Atari games, and in any case, Atari 2600 game code would have needed to perform most of the same platform-specific incantations regardless of ROM size.

We can use our results to look in more detail at the code sequences and their lineage; Figure 4 shows three of Crane's most frequently re-used sequences[5] and their relationship. The re-used sequences were fairly short overall, and a single game could be the source of multiple distinct re-used sequences.

By contrast, Carol Shaw's games exhibited no code re-use that was detected by our system – we omit the very uninteresting heat map. Five of the six games in our corpus that Shaw worked on were done at Atari, with only her final *River Raid* published by Activision. This could reflect similar factors affecting Crane's Atari work, with a learning curve or corporate development culture playing a part; it could also mean that her code re-use practice saw her making subtle, game-specific changes to re-used code that would evade detection by our system. A detailed analysis by a human analyst would need to be undertaken to answer this question more fully.

The heat map for Activision's Atari 2600 games (Figure 5) on the whole is noisier and does not enjoy the same clear results as the heat maps for Crane and Shaw. There does appear to be a clustering towards the diagonal, a preference for re-using recent code rather than older code. On the one hand, one could argue that the later code might have been improved, but it may simply have been a case of greater programmer familiarity with more recent code.

As a final application, our code re-use system can be used to help check certain claims about authorship. Recently, former Activision developer Garry Kitchen asserted on Twitter that 'A little known fact is that the Air Raid game is an illegal reskin of my code from #Atari 2600 Space Jockey' [26]. A search for matching BAD code sequences between the games reveals nine distinct matches ranging in length from 16–89 instructions. While we will not comment on the veracity of Kitchen's statement, the fact remains that our system is useful for exploring situations like this. As another example, Joe Decuir, when writing about the Atari 2600 that he helped design, recalled [16, p. 63]: 'I wrote a utility called Compute Horizontal ReSeT (CHRST) that accepted a binary value for the horizontal position and issued the hardware commands to place the object there. [...] CHRST was widely used by game designers.' For clarity, we note that the approach used by `HRCALC` is completely different than that seen in Decuir's credited games, *Combat* and *Video Olympics*. Using our system, however, we did not find any code re-use between *Video Olympics* and different games, and none of the matches found for *Combat* were related to horizontal positioning. Carol Shaw's source code for the Atari games *Polo* (1978) and *Super Breakout* (1981) contain two markedly distinct versions, `CHRST` and `CHRST1` respectively [49, 50], implying that Decuir's CHRST may be better thought of as a family of routines.

[5]The top two frequencies were distinct, and multiple sequences were tied for third place.

4 CASE STUDY: THE BREAKER PROJECT

We turn to the story of a third-party company developing Atari 2600 games in the early 1980s, in order to provide a case study of how code could be re-used both within a single company as well as draw upon code from other sources.

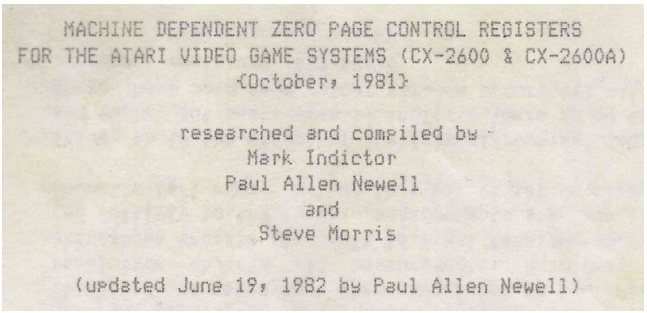
Let us begin with an explanation of our methodology. Co-author Paul Allen Newell was initially an employee of – later a consultant to – the now-defunct company Western Technologies (WT). He provides oral history here, recollections which we label with his initials (PAN) to identify, but we need not rely exclusively on oral history. Newell kept printouts of source code and other documents from his time at WT, along with a collection of 8-inch floppy disks. We had the floppy disks read by a professional data-recovery service, and then wrote our own custom program to extract not only the files from the disk images, but also the deleted files and data in unallocated file fragments. This left us with an assemblage of 487 artifacts to sift through, both physical and digital, and we are able to report the details of this case study based on documentary evidence of the time.

When WT became involved with the Atari 2600, a number of non-Atari companies were interested in producing games for the popular game console, and WT 'had a contract with [toy company] Kenner to figure out the Atari 2600 so they could make cartridges' (PAN). Unlike a console developer would today, Atari did not provide technical documentation or development kits for its 2600 to third-party developers, and was actively opposed to third-party game development [19, 52]. That means that prior to writing games for the Atari 2600, a challenging task by itself, third-party developers would need to begin by reverse-engineering the console.

The 2600's internals were sparse, and centered around three chips. The 6507 CPU was a cut-down version of the 6502 processor, with the same instruction set, and there was also a combination RAM, I/O, and timer chip from MOS Technology. Two of the three chips (and their documentation) were therefore already available outside Atari. As for software, there was no operating system on the 2600, and in fact there was no code at all except what was present in the game ROMs that were plugged into the unit. The reverse engineering efforts thus needed to be directed primarily to the third chip, Atari's custom TIA chip, and the mysteries of the 2600 were what WT's "Breaker" project set out to crack.

The project was already running in mid-1981, per PAN: 'My earliest record of Western Technologies is a first paycheck deposited on either July 2nd or 3rd of 1981. I know that on Day One I was assigned to the Breaker project that had Allen Cobb as head of project and Mark Indictor as the other programmer along with intern Steve Morris. John Hall was added to the team a couple months later.' It was fairly short-lived, and 'the Breaker project was canceled in October of 1981' (PAN) but it accomplished its goal, since WT programmers did end up producing Atari 2600 games.

Successful reverse engineering would need to yield two outcomes: infrastructure to create Atari 2600 games, and documentation for how to program the 2600. We see artifactual remnants of


MACHINE DEPENDENT ZERO PAGE CONTROL REGISTERS
FOR THE ATARI VIDEO GAME SYSTEMS (CX-2600 & CX-2600A)
(October, 1981)

researched and compiled by
Mark Indictor
Paul Allen Newell
and
Steve Morris

(updated June 19, 1982 by Paul Allen Newell)

Figure 6: Title of Western Technologies' internal Atari 2600 documentation

the former, with executables for a 6502 cross-assembler, an EPROM-burning program,[6] and 'THE BEGINNINGS OF A SOFTWARE DEVELOPMENT PROGRAM FOR THE ATARI S-100 EMULATOR SYSTEM.' What is more interesting from the code re-use point of view is the latter reverse engineering documentation. The internal manual that resulted from the Breaker project was a 20-page long document, stamped by a notary public; its title is shown in Figure 6. This manual would have been a common reference for WT programmers making Atari 2600 games.

That appears to have been the case, because assembly code from the WT manual, complete with some distinctive label names like '`FIVFTY`', was found in Newell's assembly source code. However, the WT manual was not where the code originated, which the manual makes clear: 'For the record, the sample program is borrowed from the MAGICARD book, with some additions [...] Don't want to be accused of not giving credit...'

A "MAGICARD" referred to a product by Computer Magic, Inc. (later CommaVid), and was a development system that plugged in to an Atari 2600. MAGICARD's manual contained an entire chapter devoted to explaining much of the internals of the 2600, although it stopped short of explaining the '"high resolution" graphics capability', declaring it 'beyond the scope of this manual' [15, p. 6-1], a somewhat curious decision for a manual where low-level assembly language programming was within scope. PAN recalls this external information probably being a late arrival, saying 'I realized we must have had this during the Breaker project. I think we did the bulk of our work without it, but at some point we supplemented our doc with their info.'

How might Breaker have proceeded with reverse engineering the Atari 2600 prior to MAGICARD? One obvious answer is by studying the binary code for existing published games; the 2600's game cartridges had no copy protection, and it would have been straightforward to dump the ROM contents. We see evidence that this type of code re-use happened, in fact. In the floppy disk images' deleted files and unallocated file fragments, we found pieces of a tellingly named cross-assembler listing for '`ADVENTUR.ASM`,' and this matches well to a disassembly of Atari's game *Adventure*. We emphasize that what we found is an *assembler* listing, meaning that *Adventure*'s ROM had been dumped, disassembled, and the disassembled code prepped for *re*-assembly. This was perhaps done to

[6] Idiosyncratically called the "WACKADOO EPROM PROGRAMMER, V 1.0.'

allow easy experimentation on the code when reverse engineering, and the ADVENTUR.ASM code exhibits numerous instances where addresses have been replaced by meaningful names, suggesting the code was being subjected to analysis. Indeed, PAN recalls 'dumping a lot of published games and changing the contents of registers to see what happened,' which is borne out by the WT manual referring to performing tests as part of the Breaker project.

We knew of some other re-used code in WT games from our code re-use analysis system, specifically the HRCALC code mentioned earlier. The near-final source code for *Entombed* has the HRCALC routine by the name ACTPOS – presumably "ACT" for Activision – and a comment beside it says 'ROUTINE FROM THE BOOK' that implies another in-house programming resource, since this is not present in the Breaker manual. Whether this was a literal published book or a more figurative reference is unknown; PAN does not remember a book during his time there.[7] The original source for HRCALC/ACTPOS may have been Activision's *Tennis*, because the documented source code for an unreleased game of Newell's mentions 'This is the logo section. It is the only section in any program that is a direct rip-off of someone else's game (ACTIVISION TENNIS).'[8] Here, as with HRCALC, this code re-use again involves infrastructure code, and should not be seen to detract from the originality elsewhere in the game's design and implementation.

We also see code re-use practices involving code from within WT or, as PAN characterized it, 'the original Breaker programmers shared code and tricks like fury.' The buggy pseudo-random number generator (PRNG), code re-use found by manual analysis in previous work [6] we can study from the source code point of view using Newell's artifacts. The PRNG code is accompanied in one file by the comment 'PHIL'S RANDOM NUMBER GENERATOR (SEE "RANDOM.ASM" –OR APPROPIATE [sic] DOC FILE) FOR A DESCRIPTION OF HIS GENIUS' meaning that routines existed in individual files complete with separate documentation.[9] This interpretation is bolstered by a later comment: 'THIS IS AN EARLY VERSION OF WHAT IS NOW MY "PADDLE" MODULE AND FURTHER NOTES ON SUCH CAN BE FOUND IN THE DOCUMENTED VERSION OF THAT FILE,' and PAN adds that "Paddle" was a collision detection test. The overall impression is that the Breaker programmers thought of these common routines as code to re-use in a modular fashion. To understand whether the code re-use practices at WT and elsewhere were typical, we shift to a field that has extensive experience with humans and technology: archaeology.

5 ON HUMANS AND TECHNOLOGY

In the same decades that saw the development of the Atari 2600 and its games, there were significant shifts in archaeological thought pertaining to technological processes including the examination of use and re-use. Broadly, in the 1970s and 1980s, anthropological and archaeological theoretical approaches to the study of technology and technological organization developed that can assist our understanding of the processes and mechanisms that underlie re-use. Technological organization is the study of 'the selection and integration of strategies for making, using, transporting, and discarding tools and the materials needed for their manufacture and maintenance' [34, p. 57], where emphasis is placed on understanding the dynamics of technological behavior – the dialectical interrelations of economic, social, functional, environmental, and behavioral variables of social structure manifest as and in material culture. Our archaeogaming approach is informed by this theoretical framework; we seek to understand not just how code is generated, used, and re-used but also the underlying behavioral and cultural dynamics that shape both the decisions made and the artifacts that resulted from those decisions.

All societies practice resource conservation to varying degrees, and re-use is one of the simplest and most widespread of these strategies [47]. Re-use processes occur when, after a period of use, there is a shift in the user or the activity of use for an object [46]; these include recycling (old item transformed/remanufactured into a new item), lateral cycling (unmodified old item is used in a different activity), and conservation/collecting (change in the use but not form of an old item with the intention of preservation). Understanding how these processes operate within technological systems considers both the mechanisms for acquiring the object and transferring it between individuals but also the strategies employed for procurement as either embedded (or not) in other activities and how these mechanisms and strategies are shaped by other cultural processes. In other words, *the re-use of code is not novel nor specific to game technologies; it is an expected practice within any technology.*

We can gain further insights into the possible explanations for code re-use by examining why artifacts are re-used in other human technologies. Here we draw upon the abundance of research on our oldest technology – stone. With lithic (stone) technologies, it is recognized that artifact forms and assemblage composition are the consequences of the different ways of organizing technology through the implementation of different technological strategies [34]. Generally, there are two recognized technological strategies that are relevant to discussions of re-use: curation and expediency.

Binford [8–11] introduced the concept of curation, which is 'a strategy of caring for tools and toolkits including advanced manufacture, transport, resharpening, rejuvenation, and storage/caching' [34, p. 62]. The critical variable that distinguishes curation from expediency is the advanced preparation of raw materials in 'anticipation of inadequate conditions (materials, time, or facilities) for preparation at the time and place of use' [34, p. 63]. In general, the more energy that is expended in the acquisition and manufacturing of the tool, the more likely the object is to be transported or curated [37].

Expediency, by contrast, refers to minimized technological labor (time and energy expenditure) under conditions where time and place of tool use are highly predictable [12, 34, 38]. Whereas curation anticipates the need for materials and tools, expediency anticipates the presence of sufficient materials, the absence of time stress, and longer occupation or re-use of a location to take advantage of raw material stockpiling or local abundances [34, 55]. Expedient tools are made for immediate use [8], exhibit minimal specificity in design, and are not readily maintained.

[7] For context, PAN clarifies that 'a different group of programmers did the later WT 2600 games; after the Breaker project was canceled, Mark, John, and I worked on the Vectrex project and all left the company after that (except for my consulting to finish *Towering Inferno*).'

[8] This is not the only documented admission of Activision code use. A third-party Atari 2600 programming manual completely different from the efforts described in this section contained the HRCALC code with the tagline 'Note: This program is from an Activision Game Program.' [44, p. 133].

[9] Unfortunately, PAN does not recall Phil's last name.

Although classifying an assemblage as curated or expedient is an oversimplification, these concepts are useful in describing important aspects of technological behavior [7]. Nor are curation and expediency mutually exclusive strategies – they can occur simultaneously depending on additional technological constraints and conditions [34]. But what do curation and expediency have to do with code re-use? Code is a tool. If we consider the context of game design and production at the time in question, we can see how code re-use is part of an curated strategy in response to conditions where raw materials are sparse and inadequate time is available for production.

Code is a tool, but the nature of code equally makes it a raw material. Code was initially a sparse raw material simply because it was part of an emerging technology; programmers had to use what resources were available. Writing code by hand for the Atari 2600 was intensive and expensive in terms of time, labor, and cost. Re-using code would allow for greater efficiency, and the reduction of time and effort costs associated with re-use would be preferred over generating "new" code to serve the same or similar function. Later re-use would be driven not by scarcity but by other technological constraints as the conditions that led to scarcity had been eliminated, i.e., a growing corpus of code and knowledge were available.

It is also useful to examine the role of design considerations in explaining code re-use. Design considerations, the 'variables of utility that condition the form of tools' include reliability, maintainability, versatility, flexibility, and longevity [34, p. 66].

Reliability and maintainability are the two most important design considerations that influence the lithic production technique used, as they are the determining features for whether or not a curated or expedient strategy will be employed. These are familiar concepts to programmers, and they are useful for establishing curation as underlying game production and code re-use. The criteria for identifying reliable tools or systems includes good craftsmanship [12], and certainly code selected for re-use is reliable in that it has already demonstrated that it consistently achieves the desired outcomes prior to inclusion in a new game. Maintainable strategies are also not foreign to programmers, and include simplicity of design, easy maintenance by people with poor (lithic) skills, and use in a range of functions [22]; the latter we could call versatility.

While versatility refers to the number of uses a tool is designed for [34, 51], flexibility refers to changes in tool form for different uses [22, 34, 51]. Code is a plastic raw material, much more so than stone, and is both versatile enough to serve a number of different functions but also flexible enough to adapt to the task at hand through rejuvenation and retouch if necessary. The rejuvenation and retouch of code is similar to that of stone tools – the code is reworked and parts of it are replaced to reflect the new context of use.

Longevity, or use life, is an important consideration, tied closely to curation. Re-use of code is an effective measure of the longevity of the code. There is a clear correlation between useful lifetime and the manufacture time of an object, and longevity is an interesting variable because the 'longer the use-life expectancy of an artifact, the more appropriate the artifact becomes for carrying social information' [21, p. 94].

Finally, lessons from the examination of the archaeological record caution us against considering only the function of tools especially when explaining re-use. Style also is a key consideration, and it is especially telling of the technological choices made by the tool user, although it can be challenging to distinguish function from style. Further, code re-use at the functional level does not exclude the possibility of code re-use at a stylistic or artistic level. For example, we know from artifactual evidence that Newell re-used some maze algorithm code producing distinctive mazes in an unpublished game and two published games [35].

6 CONCLUSION

Our system has allowed us to perform a large-scale study of code re-use in Atari 2600 games, discovering that there was indeed code re-use at large, as well as within the more limited scope of individual game authors and companies. In future, we could see expanding this work to have closer examinations of different game types, expanding our understanding of game culture and its development in a large corpus.

By employing our approach based on game code and saved artifacts, we have gone directly to the primary sources. Where oral history plays a role is in filling in otherwise-uncaptured background information, and we have used that in our look at code re-use within a specific game company of the era. It is important to stress that we have not singled out Western Technologies in our case study because it exhibited unusual code re-use practices. Rather, given the prevalence of `HRCALC` and other code across games, we conjecture that the practices at Western Technologies were typical of the time; we simply happen to have unique, detailed insight into this particular company thanks to Newell's oral history and artifacts.

This is where our interdisciplinary archaeogaming approach demonstrates its utility – we are able to approach code, and by extension games, as cultural artifacts. We have considered the broader culture of game development, analyzed the assemblage of code (our corpus of games), and applied lessons from the archaeological study of technological organization to understand not only why code re-use occurs but also the conditions that both created and supported this strategy.

ACKNOWLEDGMENTS

This work is supported in part by the Government of Canada's New Frontiers in Research Fund (NFRFE-2020-00880). Thanks to Andrew Reinhard and Megan von Ackermann for asking the question that led to this work, and to John Hall and Mark Indictor for their input on Section 4. We are grateful to the Strong National Museum of Play for access to the Carol Shaw and Jerry Lawson collections.

REFERENCES

[1] M. S. Ackerman and G. Parker. 18 November 1986. Process for Displaying a Plurality of Objects on a Video Screen. United States Patent #4,623,147.
[2] N. Altice. 2015. *I AM ERROR: The Nintendo Family Computer / Entertainment System Platform.* MIT Press.
[3] D. Andriesse, X. Chen, V. van der Veen, A. Slowinska, and H. Bos. 2016. An In-Depth Analysis of Disassembly on Full-Scale x86/x64 Binaries. In *Proceedings of the 25th USENIX Security Symposium.* 583–600.
[4] J. Aycock and K. Biittner. 2019. Inspecting the Foundation of *Mystery House. Journal of Contemporary Archaeology* 19, 2 (2019), 183–205.
[5] J. Aycock and K. Biittner. 2020. LeGACy Code: Studying How (Amateur) Game Developers Used Graphic Adventure Creator. In *15th International Conference on*

the Foundations of Digital Games. Article 23, 7 pages.
[6] J. Aycock and T. Copplestone. 2019. Entombed: An archaeological examination of an Atari 2600 game. *The Art, Science, and Engineering of Programming* 3, 4 (2019), 33 pages. https://doi.org/10.22152/programming-journal.org/2019/3/4
[7] D. B. Bamforth. 1986. Technological Efficiency and Tool Curation. *American Antiquity* 51, 1 (1986), 38–50.
[8] L. R. Binford. 1973. Interassemblage Variability– the Mousterian and the "Functional" Argument. In *The Explanation of Culture Change: Models in Prehistory*, C. Renfrew (Ed.). Duckworth, 227–254.
[9] L. R. Binford. 1977. Forty-seven Trips: A Case Study in the Character of Archaeological Formation Processes. In *Stone Tools as Cultural Markers: Change, Evolution and Complexity*, R. V. S. Wright (Ed.). Humanities Press, 24–36.
[10] L. R. Binford. 1979. Organization and Formation Processes: Looking at Curated Technologies. *Journal of Anthropological Research* 35 (1979), 255–273.
[11] L. R. Binford. 1983. *Working at Archaeology*. Academic Press.
[12] P. Bleed. 1986. The Optimal Design of Hunting Weapons: Maintainability or Reliability. *American Antiquity* 51, 4 (1986), 737–747.
[13] Christian Collberg and Jasvir Nagra. 2010. *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection*. Addison-Wesley.
[14] C. C. Colton. 1837. *Lacon: or Many Things in Few Words*. Longman, Orme, Brown, Green, & Longmans.
[15] Computer Magic, Inc. 1981. MagiCard Instruction Manual. https://atariage.com/software_page.php?SoftwareLabelID=281. Accessed 27 March 2022.
[16] J. Decuir. 2015. Atari Video Computer System: Bring Entertainment Stories Home. *IEEE Consumer Electronics Magazine* 4, 3 (2015), 60–66.
[17] L. M. Dennis. 2016. Archaeogaming, ethics, and participatory standards. *SAA Archaeological Record* 16, 5 (2016), 29–33.
[18] P. Deutsch and J-L. Gailly. 1996. ZLIB Compressed Data Format Specification version 3.3. RFC 1950.
[19] T. Donovan. 2010. *Replay: The History of Video Games*. Yellow Ant.
[20] M. R. Farhadi, B. C. M. Fung, P. Charland, and M. Debbabi. 2014. BinClone: Detecting Code Clones in Malware. In *8th International Conference on Software Security and Reliability*. 78–87.
[21] J. M. Gero. 1989. Assessing Social Information in Material Objects: How Well Do Lithics Measure Up? In *Time, Energy and Stone Tools*, R. Torrence (Ed.). Cambridge University Press, 92–105.
[22] B. Hayden, E. Bakewell, and R. Gargett. 1996. World's Longest-Lived Corporate Group: Lithic Analysis Reveals Prehistoric Social Organization near Lillooet, British Columbia. *American Antiquity* 61 (1996), 341–356.
[23] R. N. Horspool and N. Marovac. 1980. An approach to the problem of detranslation of computer programs. *Comput. J.* 23, 3 (1980), 223–229.
[24] Y. Hu, Y. Zhang, J. Li, and D. Gu. 2017. Binary Code Clone Detection across Architectures and Compiling Configurations. In *IEEE 25th International Conference on Program Comprehension*. 88–98.
[25] S. Hugg. 2016. *Making Games for the Atari 2600*. CreateSpace.
[26] G. Kitchen (@kitchengarry). 15 December 2021. Tweet. https://twitter.com/kitchengarry/status/1471158746302713860?s=20&t=vJgYcAH4GGoSimgcv2owBQ
[27] T. Lapetino. 2016. *Art of Atari*. Dynamite Entertainment.
[28] J. Lendino. 2018. *Adventure: The Atari 2600 at the Dawn of Console Gaming*. Ziff Davis.
[29] M. Li, X. Chen, X. Li, B. Ma, and P. M. B. Vitányi. 2004. The Similarity Metric. *IEEE Transactions on Information Theory* 50, 12 (2004), 3250–3264.
[30] L. Luo, J. Ming, D. Wu, P. Liu, and S. Zhu. 2017. Semantics-Based Obfuscation-Resilient Binary Code Similarity Comparison with Applications to Software and Algorithm Plagiarism Detection. *IEEE Transactions on Software Engineering* 43, 12 (2017), 1157–1177.
[31] U. Manber and G. Myers. 1990. Suffix Arrays: A New Method for On-Line String Searches. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*. 319–327.
[32] A. L. McDivitt. 2020. *Hot Tubs and Pac-Man: Gender and the Early Video Game Industry in the United States (1950s–1980s)*. De Gruyter Oldenbourg.
[33] N. Montfort and I. Bogost. 2009. *Racing the Beam: The Atari video computer system*. MIT Press.
[34] M. C. Nelson. 1991. The Study of Technological Organization. In *Archaeological Method and Theory. Vol. 3*, M. B. Schiffer (Ed.). Academic Press, 57–100.
[35] P. A. Newell, J. Aycock, and K. M. Biittner. 2022. Still Entombed After All These Years: The continuing twists and turns of a maze game. *Internet Archaeology* 59 (2022). https://doi.org/10.11141/ia.59.3
[36] M. Z. Newman. 2017. *Atari Age: The Emergence of Video Games in America*. MIT Press.
[37] G. H. Odell. 1989. Summary of Discussions. In *Alternative Approaches to Lithic Analysis*, D. O. Henry and G. H. Odell (Eds.). American Anthropological Association.
[38] W. J. Parry and R. L. Kelly. 1987. Expedient Core Technology and Sedentism. In *The Organization of Core Technology*, J. K. Johnson and C. A. Morrow (Eds.). Westview Press, 285–304.
[39] A. Reinhard. 2015. Excavating Atari: Where the Media was the Archaeology. *Journal of Contemporary Archaeology* 2, 1 (2015), 86–93.
[40] A. Reinhard. 2018. *Archaeogaming: An Introduction to the Archaeology in and of Video Games*. Berghahn.
[41] A. Reinhard. 2021. Archeology of Abandoned Human Settlements in *No Man's Sky*: A New Approach to Recording and Preserving User-Generated Content in Digital Games. *Games and Culture* 16, 7 (2021), 855–884.
[42] A. Sæbjørnsen, J. Willcock, T. Panas, D. Quinlan, and Z. Su. 2009. Detecting Code Clones in Binary Executables. In *Proceedings of the 18th International Symposium on Software Testing and Analysis*. 117–127.
[43] H. Sajnani, V. Saini, C. K. Roy, and C. Lopes. 2021. SourcererCC: Scalable and Accurate Clone Detection. In *Code Clone Analysis: Research, Tools, and Practices*, K. Inoue and C. K. Roy (Eds.). Springer, 51–62.
[44] San Jose Micro Technology. 1982. Atari Video Computer System Programming Manual (Revision C). Gerald Lawson papers, Strong National Museum of Play.
[45] F. Sanglard. 2018. *Game Engine Black Book: Wolfenstein 3D* (2nd ed.). Independently published.
[46] M. B. Schiffer. 1976. *Behavioural Archaeology*. Academic Press.
[47] M. B. Schiffer, T. E. Downing, and M. McCarthy. 1981. Waste Not, Want Not: An Ethnoarchaeological Study of Reuse in Tucson, Arizona. In *Modern Material Culture: The Archaeology of Us*, R. A. Gould and M. B. Schiffer (Eds.). Academic Press, 67–86.
[48] C. Shaw. 10-Feb-83. River Raid source code printout. Carol Shaw papers, Strong National Museum of Play.
[49] C. Shaw. 13-Dec-78. Polo source code printout. Carol Shaw papers, Strong National Museum of Play.
[50] C. Shaw. 15-Jul-80. Super Breakout source code printout. Carol Shaw papers, Strong National Museum of Play.
[51] M. J. Shott. 1986. Technological Organization and Settlement Mobility: An Ethnographic Examination. *Journal of Anthropological Research* 42 (1986), 15–51.
[52] A. Smith. 2020. *They Create Worlds: The Story of the People and Companies That Shaped the Video Game Industry, Volume I: 1971–1982*. CRC Press.
[53] F. Smith Nicholls. 2021. Fork in the Road: Consuming and Producing Video Game Cartographies. In *Return to the Interactive Past: The Interplay of Video Games and Histories*, C. E. Ariese, K. H. J. Boom, B. van den Hout, A. A. A. Mol, and A. Politopoulos (Eds.). Sidestone Press, 117–133.
[54] C. Therrien. 2019. *The Media Snatcher: PC/CORE/TURBO/ENGINE/GRAFX/16/-CDROM2/SUPER/DUO/ARCADE/RX*. MIT Press.
[55] R. Torrence. 1983. Time Budgeting and Hunter-Gatherer Technology. In *Hunter-Gatherer Economy in Prehistory: A European Perspective*, G. Bailey (Ed.). Cambridge University Press, 11–22.
[56] L. Wiest. 2016. Reverse engineering Star Raiders. *PoC||GTFO* 0x13 (2016), 5–20.