Experimental Archaeogaming

A Case Study

John Aycock and Katie Biittner

ABSTRACT

Archaeogaming is an area of increasing interest within archaeology. As archaeogaming's theory and practice are being fleshed out, it is worth considering if there are parallels to traditional archaeological methods within the study of video games. Here, we examine one such possibility: is there an archaeogaming equivalent to experimental archaeology? As a case study, we explore the system used for the mid-1980s development of an unreleased video game prototype for the game company Activision. Through examining this development system, whose use would be otherwise invisible in the finished software artifact, we demonstrate how we have both reconstructed a seemingly lost piece of the system virtually and used this reconstruction for experiments. The methodology we describe can be applied to digital artifacts within contemporary archaeology beyond the scope of video games, and it illustrates some key differences between studying physical and digital artifacts.

Keywords: archaeogaming, experimental archaeology, digital artifact, video game, prototype

Archaeogaming es un área de creciente interés dentro de la arqueología. A medida que se desarrollan la teoría y la práctica del archaeogaming, vale la pena considerar si existen análogos de los métodos arqueológicos tradicionales dentro del estudio de los videojuegos. Aquí examinamos una de esas posibilidades: ¿existe un archaeogaming equivalente a la arqueología experimental? Como estudio de caso, exploramos el sistema utilizado a mediados de la década de 1980 para el desarrollo de un prototipo de videojuego inédito para la compañía de juegos Activision. A través del examen de este sistema de desarrollo, cuyo uso sería de otro modo invisible en el artefacto de software acabado, demostramos cómo hemos reconstruido virtualmente una pieza del sistema aparentemente perdida, y cómo hemos utilizado esta reconstrucción para experimentos. La metodología que describimos puede aplicarse a artefactos digitales dentro de la arqueología contemporánea más allá del ámbito de los videojuegos, e ilustra algunas diferencias clave entre el estudio de artefactos físicos y digitales.

Palabras clave: archaeogaming, arqueología experimental, artefacto digital, videojuego, prototipo

Archaeogaming is broadly and succinctly defined as "the archaeology both in and of digital games" (Reinhard 2018:2). As with other areas of contemporary archaeology, archaeogaming interrogates our definitions of artifact, assemblage, site, and "the past," and in practice, it has required reconsiderations of the methods and theoretical frameworks archaeologists commonly use. From its blog-based origins (Reinhard 2013), archaeogaming now has over a decade of established scholarship: extensive introductions to—and histories of—archaeogaming may be found in Rassalle (2021) and Politopoulos et alia (2023). Initially, archaeogaming was undertaken by gamer archaeologists who recognized that they were playing games in ways that were grounded in their archaeological training and knowledge. These archaeologists wanted more realistic and representative archaeology in their games and sought ways of engaging academically with these games through research and other forms of scholarship.

Over time, archaeogaming has become increasingly interdisciplinary and has expanded to include topics as varied as discourse on ethics (Graham 2020a), the use of autoethnography (Newell et al. 2022; Smith Nicholls 2021), the framing of archaeogaming as queer gaming (Smith Nicholls 2018), the study of player-generated content (Reinhard 2021; Smith Nicholls and Cook 2022), the use of agent-based modeling and AI (Graham 2020b), the examination of the value of games and archaeogaming in pedagogy (Winter 2021), the development of strategies for the preservation of intangible digital culture (Hanussek 2019), and the use of digital technologies for transmitting archaeological and traditional cultural knowledge (Cook Inlet Tribal Council 2017). Our own work has used the theoretical framework of the *chaîne opératoire* and anthropological approaches to the organization of technology along with the tools of computer science to interpret the implementation of early video games (Aycock and Biittner 2019, 2020; Aycock et al. 2022).

And yet, despite the quality and quantity of archaeogaming undertaken in the last decade, and the appearance of archaeogaming in popular culture (e.g., the Atari video game burial at Alamogordo, New Mexico; Reinhard 2015), it remains a niche

Advances in Archaeological Practice 12(2), 2024, pp. 75–85

DOI:10.1017/aap.2024.5

within the broader discipline. One of the consequences of its continued marginalization is that archaeogaming theory and practice are still under development. One way to construct the theory and practice of this field is to look for parallels with traditional archaeology to see what has not yet been tried in archaeogaming, and what these might look like when applied in an archaeogaming context. This article examines the value of experimental reconstruction in understanding the organization of digital technologies—specifically, how it can provide insights into the development and implementation of video games.

At the same time, it would be misleading to portray this work as only being relevant to video games and archaeogaming. The digital artifact we are studying happens to be one that was used for video game development, which grounds its context of use firmly within archaeogaming, but the same techniques we are describing here are potentially just as applicable to *any* situation where unknown software and hardware are encountered. Many artifacts within the scope of contemporary archaeology are digital artifacts—a trend that is only accelerating with time—and it is therefore vital that archaeologists have methods for studying and understanding them as completely as possible. It has been observed, however, that in-depth examination of digital artifacts within archaeology has been studiously ignored for the most part (Aycock 2021); in this article, we help address this oversight by showing how we conducted experimental archaeology of one such digital (archaeogaming) artifact.

Experimental archaeology is well established. Outram (2008) provides an excellent discussion of what experimental archaeology is and is not; they situate experimental archaeology within positivist science and define it as actualistic and "(re)constructive"[1]—that is, a means of testing hypotheses and investigating those "activities that *might* have happened in the past using the methods and materials that would actually have been available" via experiments (Outram 2008:2). This is aligned with Mathieu, who characterized experimental archaeology in part as "a controllable imitative experiment to replicate past phenomena" (2002:1). Much experimental work has focused on the (re)construction of ancient technologies—from studies focused on early Oldowan lithic technologies (Schick and Toth 1994; Stout et al. 2019; Toth 1985) to artifacts from the classical world about which is there much speculation, such as the Antikythera mechanism (David 2017). But it may be surprising that such work is also needed for artifacts from the recent Anthropocene. Much knowledge and many items from the scope of contemporary archaeology are now gone or endangered, though. For example, Moshenska (2016) highlights the importance of tacit, undocumented knowledge in contemporary archaeology, and in keeping with the theme of archaeogaming, a recent survey found that the vast majority of old video games were unavailable (Salvador 2023). In this article, we examine one "lost" game-related item: a tool that was used for video game development in the mid-1980s.

Finds of tools used in the process of crafting are naturally familiar within traditional archaeology. For example, in lithic analysis, experimental flintknapping uses reconstructed tools in conjunction with analyses of debitage assemblages to understand the technological production of stone artifacts (Carr and Bradbury 2010; Crabtree 1975). And, similar to the situation we present, there are instances of tools found on sites whose exact role and usage is unclear, such as some very early forays into experimental archaeology that sought to understand whether flint tools recovered from Mount Carmel were sickles and what caused the luster on them (see Spurrell [1892] and Curwen [1930, 1935], as discussed in Ascher [1961]).

Finds of tools used in the process of crafting video games will be less familiar, however. Whereas traditional experimental archaeologists collaborate with those in the materials sciences (Wisseman and Williams 2013), archaeogaming can instead look to the expertise and knowledge of those who work in the digital realm. Here, we create a functional replica of a software development system through a process of reverse engineering drawn from computer science. We then use it for behavioral experiments to further understand the development system and its constraints and affordances. But first, some background is needed to understand our experimental archaeogaming reconstruction and the role that missing piece played in video game development.

BACKGROUND

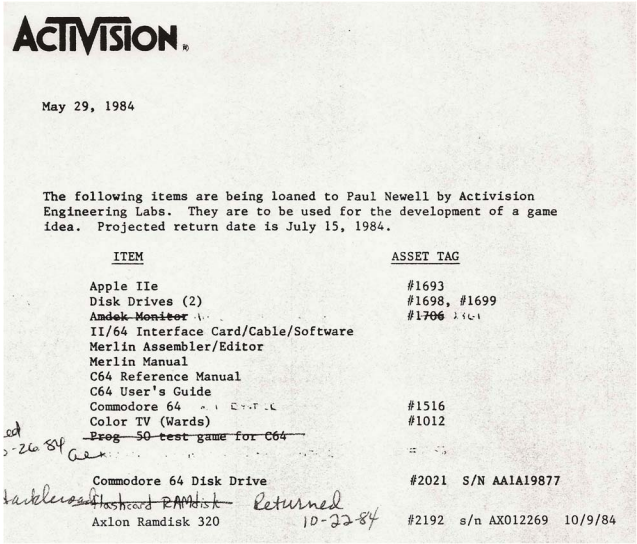
In 1984, the game company Activision commissioned a video game prototype, hiring Dona Bailey—one of the earliest female game developers—as designer.[2] Paul Allen Newell, who also had video game development experience, was in turn contracted to create the working prototype under Bailey's direction. Bailey effectively decided what to build (like an architect), and Newell realized her vision by writing computer code (the instructions the computer followed). Although the game is itself of historical interest, its examination is outside the scope of this article; instead, we focus on Newell's workflow as a professional game developer at that time period and the related artifacts.

As it happens, we have a rich assemblage of both physical and digital artifacts to work with. Newell retained printouts of code, documentation, physical correspondence, and a set of floppy disks. The disks' contents were extracted by a professional service, which yielded development-related files as digital artifacts. Newell's intuition to keep this material for so many years provides a clear documentary record and a known provenance. It is worth noting that, although the floppy disks were physical, tangible artifacts, their importance was their digital contents: the files extracted from them were not digit*ized* artifacts but born-digital artifacts. Any printout of code, in fact, captured the (possibly temporary) state of a primary digital artifact by definition.

The game prototype was being developed for the Commodore 64 computer, a relatively new machine at the time that was first released in 1982 and that went on to be a bestseller, with millions of units in circulation (Steil 2011). Newell did *not* write the game's software on a Commodore 64, however. Instead, he used a separate computer for development, the contemporaneous Apple IIe; it was physically connected to a Commodore 64 using a device that is the subject of our experimental archaeogaming, the "II/64." Normally, this whole arrangement would be unknown, barring anecdotal recollections, because it would leave no identifiable traces in the prototype game that was eventually produced, but written records document the equipment that Newell was loaned by Activision (Figure 1).

Newell's software development workflow is shown in detail in Figure 2. Newell wrote the code for the game in assembly

https://doi.org/10.1017/aap.2024.5 Published online by Cambridge University Press



ACTIVISION.

May 29, 1984

The following items are being loaned to Paul Newell by Activision Engineering Labs. They are to be used for the development of a game idea. Projected return date is July 15, 1984.

ITEM	ASSET TAG
Apple IIe	#1693
Disk Drives (2)	#1698, #1699
~~Amdek Monitor~~	~~#1706~~
II/64 Interface Card/Cable/Software	
Merlin Assembler/Editor	
Merlin Manual	
C64 Reference Manual	
C64 User's Guide	
Commodore 64	#1516
Color TV (Wards)	#1012
~~Prog 50 test game for C64~~	
Commodore 64 Disk Drive	#2021 S/N AA1A19877
~~Hashcard RAMdisk~~ Returned 10-22-84	
Axlon Ramdisk 320	#2192 s/n AX012269 10/9/84

FIGURE 1. Equipment loan documentation. (Image courtesy of Paul Allen Newell.)

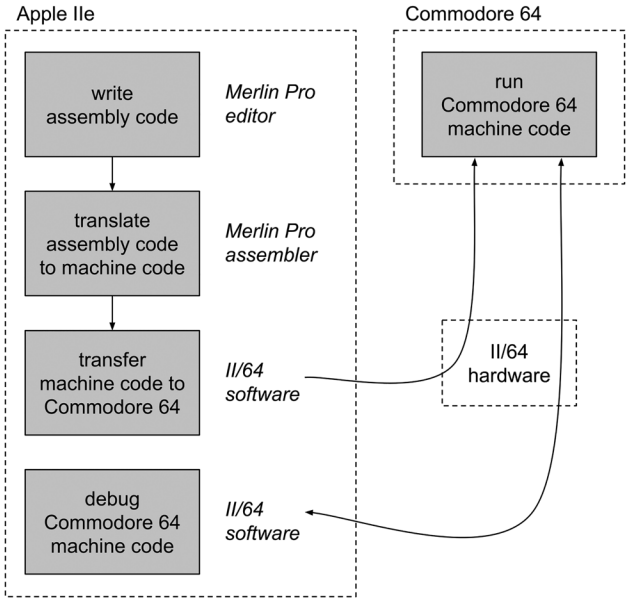


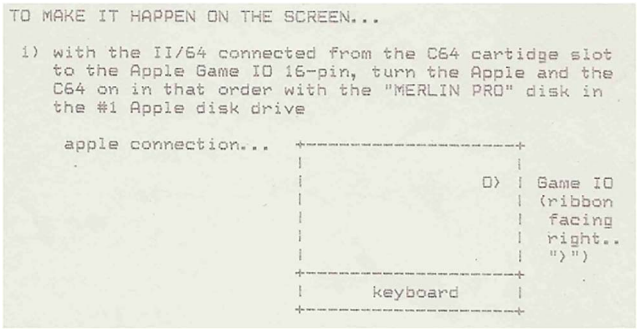
FIGURE 2. Software development workflow.

language, a "low-level" human-readable programming language that corresponds 1:1 with the binary language that the computer understands. The code editing task would have been done using the editor within the Merlin Pro software that he was loaned, which can be regarded as roughly equivalent to a rudimentary word processor.[3] Code in assembly language cannot be understood directly by a computer, and the translation from assembly to machine language was performed by a software tool called an "assembler," which was also part of Merlin Pro. Happily, the Apple IIe and Commodore 64 computers contained essentially the same central processing unit (CPU) inside the computer responsible for executing instructions. The machine language was therefore the same for both. The binary code the assembler produced could then be transferred via the II/64 from the Apple IIe's memory to the Commodore 64's memory to be run, and the code on the Commodore 64 could have been debugged from the Apple IIe using the II/64 as well. And although we could locate archived versions of the Merlin Pro manual to understand its usage and capabilities, the II/64 was much more mysterious.

Unlike many artifacts from that era of computing, we could find no information about the II/64, and as a niche product, very few of them may have ever existed. We do know that the II/64 was made by a high school student, Chip Gracey, and sold through his company Innovative Software Engineering (Parallax 2021). Thanks to the instructions Newell wrote (Figure 3), we also know the II/64 consisted of a hardware cartridge that plugged into the Commodore 64, with a ribbon cable connecting the cartridge to the Apple IIe—specifically, to the Apple IIe's game I/O connector. Some II/64 software would then be run on the Apple IIe that would be able to talk to the hardware cartridge in the Commodore 64, giving the ability to remotely control aspects of the Commodore 64 from the Apple IIe. The absent hardware cartridge likely contained some unmalleable software in "read-only" memory that the Commodore 64 would run to perform its half of the communication. For clarity, we will collectively use "II/64" by itself to refer to the system as a whole, qualifying it as "II/64 hardware" or "II/64 software" when more specificity is required.

The physical bridging of the two computers is easy to gloss over, but it is important in understanding the II/64's context of use. It is hard to appreciate, in an age of computers with USB connectors, what a delicate process it was to plug something like the II/64 hardware into the Apple's game I/O connector. Typically, that connector would be used for joysticks and other game controllers, so this was not something done exclusively by software developers; regular home computer users would have also needed to attach devices to that connector. The top of the Apple IIe's case would have to be opened (it is not screwed on, in fact, so as to facilitate internal access), exposing the static-sensitive circuitry, and a cable ending with 16 small, easily bent metal pins would need to be plugged precisely into the awkward-to-reach 16-pin game I/O connector on the computer's main circuit board (Figures 4 and 5). Luckily, this fraught operation would only need to be performed correctly once, when the II/64 was initially set up.

This two-computer arrangement for software development is not unique and is still used today in certain situations. It is referred to



TO MAKE IT HAPPEN ON THE SCREEN...

1) with the II/64 connected from the C64 cartidge slot to the Apple Game IO 16-pin, turn the Apple and the C64 on in that order with the "MERLIN PRO" disk in the #1 Apple disk drive

apple connection...

Game IO (ribbon facing right..)

keyboard

FIGURE 3. Development instructions excerpt. (Image courtesy of Paul Allen Newell.)

https://doi.org/10.1017/aap.2024.5 Published online by Cambridge University Press

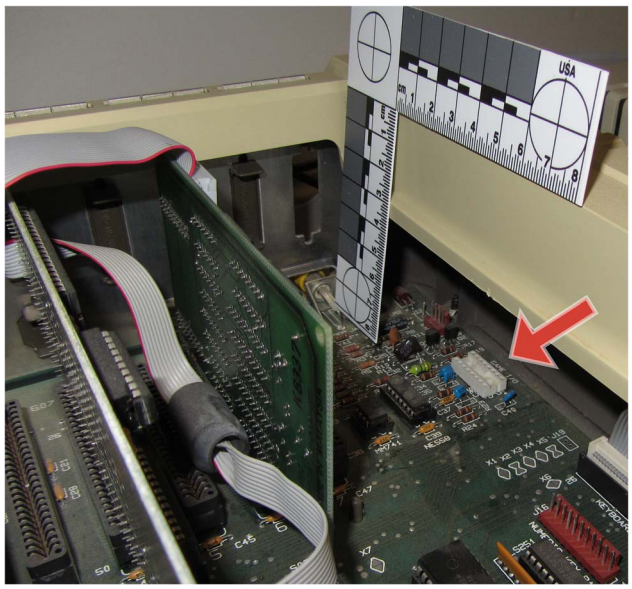


FIGURE 4. The game I/O socket inside an Apple IIe computer, indicated by the center-right arrow (digitally added). (Image courtesy of John Aycock.)

as a "cross-development system." Why would a cross-development system be used? We can look to physical analogs to understand the reasoning, such as using clay to form molds for metal casting (Bruhns 1972; Rose et al. 2023; Silas 2005), where working in one medium allows more ease and flexibility than another. If we extend this lens of ease and flexibility to the domain of computers, a cross-development system would be used when one or more of the following situations exist:

(1) The "target" computer is unable to use software development tools such as editors and assemblers in a reasonable way. For example, it would not be possible to develop code on a video game console without a keyboard, or on a console that had a paucity of memory. In our case study, this does not apply, because the Commodore 64 target system is a computer that is perfectly capable of running such tools.
(2) Software development tools do not (yet) exist for the target computer. Again, this is not true for the Commodore 64, which had the BASIC programming language built in, and Commodore had even produced its own editor/assembler software the same year the computer was released (Commodore Business Machines 1982a).
(3) It is challenging to have both the development tools and the software being developed coexist on the target computer. This situation did apply here. Although the Commodore 64's memory size was not as scarce as it was in computers that preceded it, having development tools in the computer's memory would have precluded creating the ambitiously large game Bailey and Newell had devised. Alternating between development tools and the game—completely swapping one in favor of the other—would have been possible but would have negatively impacted development time. Another consideration of home computers during that time period is that computer programs in the memory of the same computer had no protection from one another. Even if the game and development system could have both fit in the Commodore 64's memory simultaneously, a bug in the game could have crashed the computer or corrupted everything in its memory. It was advantageous for several reasons to have a separate development computer.
(4) The development computer is notably better equipped than the target computer. This applied in three different ways, based on documentary evidence. First, Merlin Pro's minimum requirements meant that the Apple IIe would have had twice the amount of memory as the Commodore 64 (Bredon 1984b). Second, Figure 1 lists an "Axlon Ramdisk 320," which acted like extremely fast floppy disk storage for the Apple (Axlon 1982), which would have sped up development. Third, again a speed enhancement, Newell's correspondence indicates that his Apple IIe contained an "Accelerator IIe"—hardware that would make the Apple's software "run approximately 3-1/2 times faster" (Titan Technologies 1984:6). Somewhat ironically, reminiscent of the Ea-nāṣir tablets (Rice 1994), we only know about the Accelerator IIe being used because Newell's letter was a complaint about his development computer crashing when the device was installed.[4]

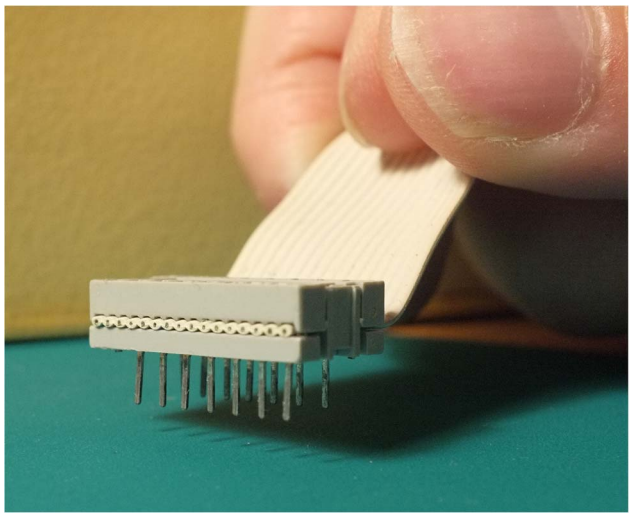


FIGURE 5. A 16-pin game I/O connector, with fingers for scale. Underscoring its delicacy, two of the pins have broken off this (joystick) connector as a side effect of handling. (Image courtesy of Hayden Kroepfl.)

With this background, we can now concentrate our attention on the II/64. The dearth of information about the II/64 leaves us with a critical gap in our knowledge of how the game was crafted. The II/64 was a conduit between the two computers, and it need not have left any distinguishing "tool marks" to speak of. However, the II/64's use would have constrained development practices by virtue of its affordances. How can something that is physically absent be reconstructed?

UNDERSTANDING AND RECREATING A DIGITAL ARTIFACT

The necessary first step toward an experimental reconstruction of a digital artifact is developing an understanding of what it does and how it functions. As a starting point, Newell's development

https://doi.org/10.1017/aap.2024.5 Published online by Cambridge University Press

instructions from that time documented how the II/64 software was started from Merlin Pro and what he typed at the II/64's textual command-line "+" prompt to get the game running on the Commodore 64. In conjunction with Commodore 64 technical documentation (Commodore Business Machines 1982b), we were able to infer what some of these II/64 commands did. Using the II/64 from the Apple IIe keyboard, Newell was able to set the value of single memory locations in the Commodore 64, transfer memory contents en masse from the Apple to the Commodore, and start the Commodore 64 running machine language instructions at a specified memory location. This may have been only a fraction of the II/64's capabilities, however.

Although we did not have the physical II/64, we had the next best thing: a digital artifact. Newell's floppy disks contained a copy of the II/64 software that ran on the Apple IIe. Consisting of just under 1,700 bytes, this relatively small binary file contained the machine code and associated data necessary for the Apple IIe to control the II/64 hardware. Reverse engineering the binary would definitely tell us something about the missing device, but it was not clear how much it would reveal. For example, the software may not have done anything on the Apple IIe apart from handing off the II/64 commands untouched to the (now-absent) software in the Commodore 64 cartridge. In that case, we would learn next to nothing about the II/64's full functionality from the digital artifact we had.

For an initial assessment, we looked through the binary file for printable strings—messages that might have been printed on the screen. Perhaps there might even be a "help" screen listing the II/64 commands. The individual symbols within a string would be represented using a numeric encoding of some kind, and given the type of computer and the time period, the most likely encoding candidate would be one called ASCII. ASCII, an abbreviation for the wordier American Standard Code for Information Interchange, is a computing standard that defines how symbols correspond to numbers. The uppercase letter "A" is represented by the value 65, for example. However, our search turned up nothing using tools that looked for printable ASCII strings, using a modern computer to examine the digital artifact of the II/64 software.[5] Apple ASCII had a slight quirk in that its ASCII values could be greater than "normal" printable ASCII values (Apple Computer 1985), and once we compensated for that, a string with an unmistakable message appeared:

][/64 SYSTEM
(C)1984 BY ISE
DB CHIP GRACEY
VERSION 1.0
SERIAL #110001

This gave us a date and attribution for the digital artifact that was consistent with other evidence. The "II" of the "Apple II" designation was often stylized using square brackets, and seeing the name given as "][/64" was not a substantive difference. There was no other clear information, and to analyze the binary further, we needed to examine the machine language code it contained.

Binary machine language code can be automatically converted into an assembly language representation using a software tool called a "disassembler," which takes advantage of the 1:1 correspondence between assembly and machine language mentioned earlier. The original assembly process, however, discarded substantial information that cannot be recovered, and what the disassembler produces is not precisely equivalent to the original; for instance, any human-readable comments the programmer placed in the original assembly code are lost in translation. The only way to rediscover knowledge about the code is through manual analysis of the reconstructed assembly language. This is a skill that is learnable, but it may also be done in interdisciplinary collaboration with computer scientists—not unlike how archaeologists would work with people in other fields for scientific analysis of physical artifacts, such as working with geneticists on ancient DNA recovery and replication from skeletal materials (Sedig 2019).

As a concrete example of how reverse engineering code can be done, we consider a discovery we made early on in the code that acted as the Rosetta Stone for understanding how the II/64 communicated between the Apple IIe and the Commodore 64. Because the II/64 hardware only connected to the Apple IIe via its game port, logically, all communication had to flow through there. Thinking of that port's potential use with joysticks, one would expect the ability to send input signals into the Apple IIe. What is unexpected is that the game port had "five output signals" as well (Apple Computer 1985:190). Moreover, the Apple IIe employed what is called "memory-mapped I/O," meaning that special memory addresses directly manipulated those input and output signals. Any II/64 code that accessed those special addresses had to have been part of the Apple–Commodore communication, which therefore provided us with digital wayfinders in the assembly code. One of those code sequences was especially interesting.

This code sequence, in the span of five consecutive instructions, did five things in the following order:

(1) Set output signal #1 to the value 0;
(2) Set output signal #2 to the value 1;
(3) Set output signal #3 to the value 0;
(4) Set output signal #4 to the value 1;
(5) Set output signal #4 (again!) to the value 0.

What is critical to remember is that the II/64 connected two completely independent computers. The last two steps—changing output signal #4 to 1 and then to 0 immediately afterward—could happen so quickly that any II/64 software running on the Commodore 64 might easily overlook it . . . unless the important effect was not the values at all but the act of *transitioning* from a 1 to a 0.

The type of CPU present in both the Apple IIe and the Commodore 64 had the ability to be interrupted, for a signal occurring at an arbitrary time to force the CPU to run different code to handle the event, much as the chime announcing an incoming text message might cause a person to interrupt their current task to attend to the text. Of those interrupts, one had the property of being what is technically called "negative edge triggered" (MOS Technology 1976)—in other words, the interruption would be caused by a transition of that signal from a 1 to a 0. Moreover, that particular interrupt signal, the so-called nonmaskable interrupt, was accessible to a cartridge plugged into the Commodore 64 (Commodore Business Machines 1982b) just as the II/64 was. Putting all this together, we surmised that the II/64 hardware had the Apple's output signal #4 connected to the

https://doi.org/10.1017/aap.2024.5 Published online by Cambridge University Press

Commodore 64's nonmaskable interrupt line, and the above code sequence was setting a value of 010 to be read using the first three steps, with the last two steps causing an interrupt on the Commodore 64 to make it pay attention to the incoming 010 value. Determining the significance of the value 010 was then a follow-on task for further reverse engineering.[6]

Another two starting points for reverse engineering the code came thanks to Newell's development instructions, which showed both how the II/64 software was first loaded into the Apple IIe's memory (causing some II/64 code to be run) and how that in-memory II/64 code was invoked later. The latter code led almost immediately to the area of the II/64 code that input and interpreted user commands. Using that code, we could determine the full set of II/64 commands and, from there, what each one did. As we guessed, the II/64 did support more than Newell's documentation happened to capture.

Some of the II/64's commands were local to the Apple IIe only, such as a command to leave the II/64 prompt and return to the Merlin Pro menu. The purpose of these local commands was found by analyzing the II/64 code and, given that they did not require a physical II/64, the analysis was verified by trying out the commands in a software-based Apple IIe emulator. The more interesting commands, however, involved Apple–Commodore interaction, because they would be ultimately responsible for facilitating or limiting development practices. We knew three of these already from Newell's documentary evidence, at least in part. Where Newell had only needed to record how to transfer memory contents from the Apple IIe *to* the Commodore 64, we learned from the code that the memory transfer could also be done in the other direction. And, moreover, selected memory regions of the Commodore 64 could be viewed in two ways: either as a numeric view or in a disassembled view of the memory contents interpreted as assembly instructions. A separate "T" command (sTatus?) would return the current contents of the Commodore 64's CPU registers, which were locations for storing values that were separate from the computer's memory proper. All told, these features for remotely interrogating the Commodore 64's state would have been invaluable for debugging.

In addition, a plausible use for the fifth output line in the Apple IIe's game I/O port was found, which had not previously been accounted for. The II/64 "RS" command would cause that output to be repeatedly set to 0 for an extended period, and we suspect that, given the command name, the II/64 hardware connected this to the Commodore 64's ReSet signal to restart the remote computer.

The design of the II/64 software on the Apple IIe was effectively in three layers (Figure 6), which can be thought of as strata in the software. The topmost user interface (UI) layer is what the user of the II/64 interacted with. With the exception of the ReSet command, which manipulated an output line directly, all the other UI-layer commands addressing the Commodore 64 were implemented by the second layer, which we called the "command layer." The command layer was only able to issue four commands: read from Commodore 64 memory, write to Commodore 64 memory, run code starting at a given location, and return the CPU register contents. That means that no matter how many UI-layer features there were, ultimately, they were all restricted in that they could only interact with the Commodore 64 in terms of these four

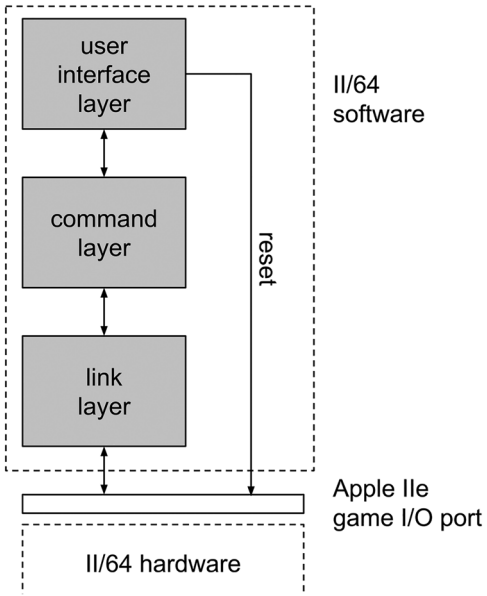


FIGURE 6. Conceptual layers within the II/64 software on the Apple IIe.

commands. A further complication is that the four commands were expressed in bytes: a grouping of data where a single byte consists of eight bits, and an individual bit can be 0 or 1. The problem is that there were not enough input and output lines available to the II/64 to express an eight-bit byte all at once, and that was the task of the link layer: communicating data to and from the Commodore 64 in piecewise fashion, two bits at a time. In fact, what we saw in the code sequence outlined earlier was the start of the link layer indicating to the Commodore 64 that a command was forthcoming.

With analysis complete, we could begin experimenting. First, we were able to verify how the II/64 commands appeared to the user by essentially hotwiring the II/64 code. Where the link layer expected a response from the missing II/64 hardware, we modified that code by changing a mere 10 bytes so that the communication always seemed to succeed. Our modified version of the II/64 software, when run in a software-based Apple IIe emulator (MAME 0.255), allowed us to try all the II/64 commands, although the "data" reported from the fictitious Commodore 64 always consisted of zeroes. But we could improve on this.

For our second experimental system, we wrote a Python program that mediated between (1) the original, unmodified II/64 software running in an Apple IIe emulator and (2) a virtual Commodore 64 running in a different software emulator (VICE 3.7). Our program behaved as our analysis suggested the missing II/64 hardware behaved, thereby allowing us to interact fully with the II/64 as Newell once did. A screenshot of this experimental system running is shown in Figure 7. The commands in the Apple IIe window include transferring data to display "HELLO SAA" to the Commodore 64's (screen) memory, disassembling data from the Commodore's memory, displaying a portion of that same data as numbers, and requesting the Commodore's CPU register values. We have recreated the II/64.

https://doi.org/10.1017/aap.2024.5 Published online by Cambridge University Press

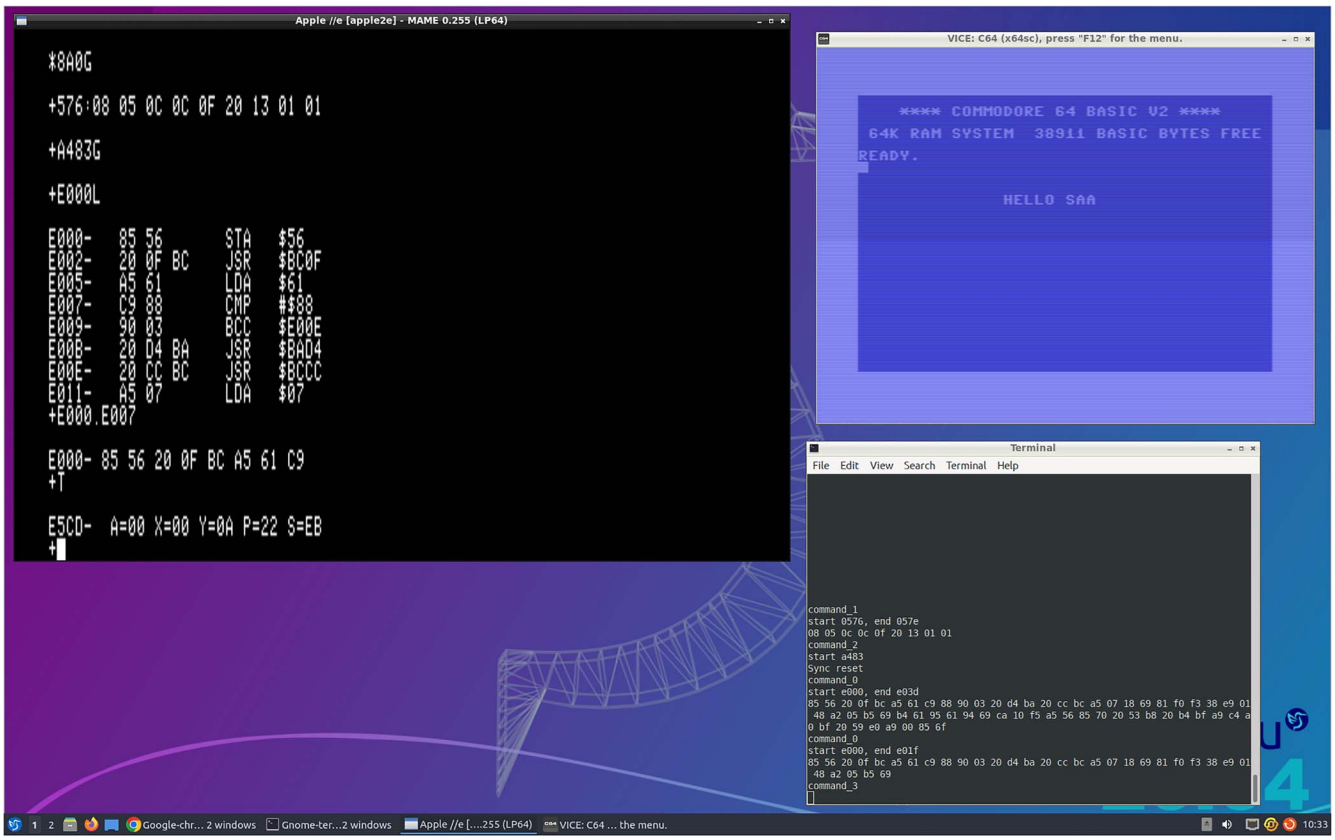


FIGURE 7. Reconstruction screenshot. Windows clockwise from upper left: Apple IIe emulator, Commodore 64 emulator, II/64 reconstruction program.

ARTIFACT AS POLYGLOT

Digital artifacts may be viewed as a form of language(s). Considering the II/64 through a linguistic lens, we have already seen assembly and machine language, which are alternate representations of the same code, albeit one for humans and one for the computer. Furthermore, each conceptual layer inside the II/64 software implements a different language: from the language the UI layer provides to the user, to the restricted four-command language of the command layer, to the physically constrained bitwise language of the link layer. And, as we discovered through our analysis of the digital artifact, there was yet one more language.

There was some code in the II/64 software whose purpose was unclear during our initial reverse engineering, until we read carefully through the Merlin Pro manual. Merlin Pro had the ability for other software to extend its set of assembly language commands in a controlled manner (Bredon 1984b), and the II/64 software had taken advantage of this. The II/64 extensions to the Merlin Pro assembly language, based on our analysis, would have greatly sped up the development workflow—for instance, machine code could be transferred to the Commodore 64 and run automatically as a side effect of running the Merlin Pro assembler. With our experimental reconstruction of the II/64, we were in an ideal situation to test this, and we discovered something odd: half of the extension language commands did not work.

Our reconstruction gave us a vantage point on the II/64 that was very different from the original hardware, one where we could watch the data being sent back and forth along the II/64, and we noticed that the bytes the faulty language extensions sent to the Commodore 64 were wrong. The problem was neither a bug in our reconstruction (our first thought) nor a bug in the II/64 software. What we realized is that the II/64 software was not designed to work with Merlin Pro, but instead the earlier version of the software that was simply called Merlin. Merlin Pro made a change to the location of some crucial data in the Apple's memory compared to Merlin (Bredon 1984a, 1984b), which was why the II/64 software was sending the wrong data. Thanks to our reconstruction, we were able to test this hypothesis by pairing the earlier version of Merlin with Newell's II/64 software, and the II/64 extension language worked correctly as predicted.

THE LIMITS OF RECONSTRUCTION

Experimental archaeology cannot address all questions about physical artifacts, and the same is true for digital artifacts. Consequently, there are some aspects of the missing II/64 hardware that we are unable to reconstruct with complete assurance. Although we can know what data flowed through the four command-layer commands based on its usage in the II/64 software on the Apple IIe, we do not completely know what the software inside the Commodore 64 cartridge did. For example, we

https://doi.org/10.1017/aap.2024.5 Published online by Cambridge University Press

inferred that the "T" command meant sTatus based on the data returned, but it may also have been sTep, a powerful debugging facility that would permit executing a single instruction at a time. Technically, the computer's hardware did not support this, but it could be achieved in software, as an earlier model of the Apple II demonstrated (Apple Computer 1978). Both sTatus and sTep could return the same data, so we cannot determine the correct interpretation without additional evidence.

Another key question is whether the Commodore 64 continued to run a program after a II/64 command had been issued from the Apple IIe, or whether it was effectively paused to await more II/64 commands. Either arrangement would have been possible, but interacting with a remote computer that is actively running the program being debugged would require the programmer to work in a different way.

The good news is that software is extremely malleable: because we reconstructed the system in software and not hardware, we can change the behavior of these aspects by simply changing our program's settings and re-running it. This allows us the ability to experiment with a range of possible configurations to understand their effects.

DISCUSSION

Because the description of our work is undeniably technical in a way that archaeological readers may be unfamiliar with yet, it is helpful to begin the discussion with a recap. We have interpreted the "experimental" of experimental archaeology in two ways. First, we have created what Mathieu (2002) would term a "functional replica" of the II/64's missing components: its hardware and the related software that would have resided in the II/64's hardware cartridge. Our reconstruction is validated both by the fact that we can successfully follow Newell's contemporary instructions involving the use of the real, original II/64, as well as our ability to use even those II/64 commands that Newell did not capture in his documentation. Second, we applied our reconstruction to perform behavioral experiments; importantly, these involved the use of the real digital artifact that remains, the II/64 software for the Apple IIe. Through these experiments, among other things, we discovered a bug in the II/64's operation and were able to test our hypothesis about why this occurred. We also saw limitations in the affordances the II/64 provided to the programmer—for example, the programmer could view the Commodore 64's CPU registers but could not change them.

The methodology we used to analyze the II/64's remaining software in order to build our replica was reverse engineering the binary code and data in the digital artifact, and we have provided some key illustrative examples of how that was done. Reverse engineering a digital artifact is a technique that draws on our interdisciplinary archaeology / computer science collaboration, but there are established ties to archaeology in general and archaeogaming specifically. Moshenska (2016) talks about the linkage between reverse engineering and contemporary archaeology, and our past archaeogaming work—although not focused on experimental archaeology—has used reverse engineering to study video games individually and at scale (Aycock and Biittner 2019, 2020; Aycock et al. 2022). A recent book has further shown how the process of reverse engineering a digital artifact can be seen to parallel Carver's archaeological field research procedure (Aycock 2023; Carver 2009).

There is a temptation to see Newell's use of the II/64 for software development as idiosyncratic and therefore not representative of a larger class of human behavior, but this is not the case. Cross-development systems are still in use today, as we noted, and our work gives us insight into the early use of such a system in video game development. The documentation in Figure 1 confirms that the II/64 and other development paraphernalia were supplied by the game company Activision to Newell for the purposes of development. This was not a system that Newell already possessed, and he confirmed that this system was one of Activision's choosing—not something he had specially requested. It is not unreasonable to infer that similar cross-development systems were used by other Activision developers at the time. There are anecdotal accounts of in-house cross-development systems being used elsewhere in early video game development targeting home computers (e.g., Pape 2013:47–48; Taylor 1984), but by leveraging Newell's assemblage, we have been able to explore this early cross-development system in depth.

In many ways, the experimental archaeogaming approach we have described here is akin to that of experimental work on the lost-wax casting technique. There are many experimental studies focused on the lost-wax (or *cire perdue*) casting technique (including Ascher 1961; Congdon 1985; Daragan and Romanenko 2021; Goren 2008, 2014; Rose et al. 2023; Wang et al. 2023). If we look at a very simplified operational sequence for lost-wax casting, first a wax model is produced, then a clay mold is formed around the wax model. Heating the mold causes the wax to run out, and the resultant hollow is filled with molten metal to produce the objective piece. To extract the metal objective piece, the mold is broken; this intentional and required destruction of the mold means that only fragments of them are recovered. Although some of the objective pieces may still have remnants of the mold or core (Rose et al. 2023), the surfaces of these objective pieces are typically finished via polishing. These finishing techniques will not only remove mold or core remnants and residues but also eliminate any seams or other attributes indicative of lost-wax casting. The loss of both the wax model and the mold means that most of the production process is absent from the archaeological record, which has necessitated experimental and ethnoarchaeological research to understand this metallurgical technology and determine the constraints of the various materials and techniques used (Rose et al. 2023). For this reason, we can use the experimental work done on the lost-wax technique as an analog for the experimental work on the cross-development system we have described here.

In archaeogaming, we have similar concerns about the invisibility of the production process in the (digital) archaeological record. As discussed above, although we may have the end product of the video game production sequence (i.e., the game), the tools used can be invisible or "lost" in the assemblages we are working with, and the physical components used in the construction of digital artifacts (and tools) may be unknown. In our case, we know from physical documentation and conversations with Newell what physical components were needed. Through reverse engineering, however, we are able to test various components within the operational sequence to determine both various constraints Newell encountered while programming the game and the

https://doi.org/10.1017/aap.2024.5 Published online by Cambridge University Press

specific constraints that resulted from working in a cross-development system that are captured neither in documentation nor in Newell's memory. Although we may not need to conduct invasive analyses to determine the composition of our materials and their sources in the same way those examining *cire perdue* would, our approach to understanding the materials used is also similar in that an interdisciplinary approach is required. This research was possible because of one author's expertise in, and access to, the physical and digital materials used and that individual's ability to reconstruct and replicate the production environment. Indeed, it is the digital immateriality of our artifacts where the core distinctions lie.

That said, the fact that we have created our functional replica using software emulators and the programming language Python—a language that did not even exist in the 1980s—may come as a surprise and might appear to invalidate our work as experimental archaeology. An authentic *material* replica, by contrast, would require connecting a physical Apple IIe to a physical Commodore 64 using reconstructed II/64 hardware built with period-appropriate electronic components. However, our approach is not only valid but serves to illustrate some of the challenges to archaeology posed by archaeogaming that were mentioned in the introduction.

The Outram quote we used earlier in defining experimental archaeology would seem to differ, ending as it does with an injunction to use "the methods and materials that would actually have been available" (Outram 2008:2). The very next sentence offers an important qualifier, though: "This is not to say that *all* materials and methods need to be authentic in experimental archaeology, but certainly those pertinent to the hypothesis" (Outram 2008:2). This is not a lone view, given that Mathieu (2002:3) states, "Functional replicas need not be authentic in every respect, only in those aspects which are deemed relevant to the object's normal functioning." On the surface, this suggests that some amount of abstraction may legitimately be applied to replication. We do not disagree. We argue the opposite, though: *our replication does not abstract anything away*. To understand the justification for this bold claim, we observe that there is an implied assumption made by Outram and Mathieu. For physical artifacts, the world the artifact exists in and the real world are one and the same; it is so obvious that it does not bear discussion. This is physically oriented thinking, and we are working with a digital artifact, not a physical one. In actual fact, for both physical and digital artifacts, *authenticity is relative to the environment in which the artifact exists*. With a physical artifact, the environment is the real world, but with a digital artifact such as ours, this is not necessarily the case. By running the II/64's software in an Apple IIe emulator with a high degree of verisimilitude, the II/64 software is effectively being run in its own, separate virtual universe. The II/64 software and, for that matter, other software running within the Apple IIe emulator cannot detect that it is not being run on real Apple IIe hardware,[7] and our II/64 hardware reconstruction in Python interacts with the emulated Apple IIe using an interface that makes it all but omnipotent with respect to the virtual Apple IIe. Consequently, relative to the (virtual, emulated) environment, our replication is perfectly authentic. An identical scenario plays out between our II/64 hardware reconstruction and the emulated Commodore 64.

Another way that archaeogaming forces reconsideration of archaeological methods is in its treatment of (digital) artifacts. Regarding the use of real artifacts in replication, Mathieu writes that, because it "requires . . . the use of a somewhat unique artifact, it is often the most expensive and least pursued form of replication" (2002:3). Again, this is physically oriented thinking. Our original digital artifact, the Apple IIe portion of the II/64 software, is backed up; even when we perform behavioral experiments that result in the emulated code changing the contents of the digital artifact, we can easily restore a pristine version of it afterward. Destructive experiments within our virtual environment cause no lasting damage. This means that our experiments are repeatable by us and reproducible by others, permitting a well-founded scientific approach to archaeological experimentation.

Putting these two ideas together, our preliminary experiment before creating our Python-based II/64 replica involved what we characterized as "hotwiring" the II/64 code by changing 10 specific bytes of it. This could have been accomplished by altering the digital artifact prior to using it in the Apple IIe emulator, a change that could be trivially undone using the digital artifact's backup copy. Instead, we took advantage of the virtual universe of the emulated Apple IIe in a different way. We allowed the II/64 software to start normally inside the emulator and become resident in the emulated Apple IIe's memory. Then, we paused the emulator: from the point of view of the emulated Apple IIe, the flow of time in its virtual universe stopped in an undetectable fashion. We could then make the necessary 10 bytes' worth of changes from our extra-universe vantage point and resume the emulator's execution. From the perspective of the emulated Apple IIe, the II/64 software was modified spontaneously and instantaneously, and it can be done repeatably.

One final note pertains to the scope of applicability of this work. It would be easy to relegate our experiments to archaeogaming, or even a narrow subinterest within the inclusive umbrella of archaeogaming, but there is much more at stake. Archaeogaming's deliberate engagement with digital artifacts and virtual environments makes it a crucible for developing techniques through which archaeology in general can address artifacts of the present and the recent contemporary past. Software development is a human activity, where programmers are creating, interacting with, and being constrained by technology. The approach we have described here is not at all limited to video games, and it provides an exemplar of ways in which digital artifacts can be studied archaeologically.

CONCLUSION

Archaeogaming is a productive and, importantly, creative endeavor that represents one pathway to addressing limitations in traditional approaches to understanding our collective past. It engages with the theory of materiality while explicitly addressing the consequences of the intangibility of digital artifacts on the methods we can use. As we have demonstrated, it is possible to analyze a digital artifact thoroughly even when we are left with only its collection of binary 1s and 0s. From this effort—conducted using an interdisciplinary approach, just as what archaeology would use for the specialized analysis of physical artifacts—we were able to learn about this behind-the-scenes software development tool, its capabilities, and its limitations. Ultimately, what the II/64 did and did not do shaped the experience of programmers and how they were able to perform their work.

https://doi.org/10.1017/aap.2024.5 Published online by Cambridge University Press

More importantly for the development of archaeogaming theory and practice, we have shown how analysis can be applied to bring a digital artifact to life again. This has allowed us to verify our analysis, experiment with the system, and even uncover some hidden flaws that might otherwise have easily been overlooked. We want to stress that, even though the missing II/64 aspects were physical, the reconstruction need not be, and this gives us substantial flexibility for experimentation with its unknown characteristics.

Although future generations of archaeologists will undoubtedly engage with both digital and material artifacts, it is *our* responsibility to continue to develop robust archaeogaming theory and practice *today*. Every day, humans spend a significant amount of their time working and playing on their devices, using digital tools, and creating huge quantities of born-digital artifacts. We face a "tsunami" of digital artifacts (Aycock 2021), so it is critical that we continue to reimagine, redevelop, and revise our ways of doing archaeology.

Acknowledgments

Many thanks to Dona Bailey and Paul Allen Newell for sharing their recollections, providing feedback on this article, and supplying the artifacts that made this work possible. Thanks also to the anonymous reviewers, whose comments were instrumental in improving this work. Carolina Avendaño Duque assisted with the Spanish-language abstract, and Jeremy Penner provided a critical hint for using the MAME Lua interface.

Funding Statement

This work is supported in part by the Government of Canada's New Frontiers in Research Fund (NFRFE-2020-00880).

Data Availability Statement

The source code for our reconstruction is freely available at https://github.com/aycock/ii64.

Competing Interests

The authors declare none.

NOTES

1. As discussed in Outram (2008:2), the use of the "re-" prefix is not universally loved. We have chosen to use the variant "(re)construction" except in cases where recovered artifacts have been replicated.
2. Background information is drawn from interviews with Dona Bailey and Paul Allen Newell, with ethics approval from the MacEwan University Research Ethics Board, file 102076. Bailey is best known for Atari's *Centipede* (1980), and Newell published games for the Atari 2600 and Vectrex, along with the arcade game *Cube Quest* (1983).
3. Although the manifest in Figure 1 says Merlin and not Merlin Pro, Newell's documentation mentions the latter (Figure 3), and we also found the Merlin Pro software on the development disks he retained. This may seem to be a minor point, but it has ramifications we discuss later.
4. We note that the Accelerator IIe is not listed in Figure 1, but then that manifest also does not list other hardware inside the Apple IIe that would have needed to be present, such as a floppy disk controller card. We conjecture that only obvious, discrete physical items were recorded in that list.
5. We used the strings program on Linux for this task. Similar programs are available for Windows and Apple's OS X.
6. The 010 turned out to be the initial part of the II/64's "handshake" between the Apple II and the Commodore 64 to verify that the Commodore 64 was attached, powered on, and responding properly.
7. The reality of emulator detection is more nuanced. Modern software emulators for retrocomputing platforms such as the Apple IIe are extremely accurate and constantly improving, but *perfectly* accurate emulation can be difficult to perform correctly. It is possible to craft a modern program that, when run within a software emulator, tries to detect obscure "edge cases" where an emulator's behavior is known to differ from that of the real physical computer (Pilgrim 2014). Having said that, the software we ran inside the Apple IIe emulator was well over 30 years old and could not have been written with foreknowledge of emulators in the distant future (in computing terms), so this possibility can be safely disregarded without loss of generality.

REFERENCES CITED

Apple Computer. 1978. *Apple II Reference Manual (January 1978)*. Apple Computer, Cupertino, California.

Apple Computer. 1985. *Apple IIe Technical Reference Manual*. Addison-Wesley, Reading, Massachusetts.

Ascher, Robert. 1961. Experimental Archeology. *American Anthropologist (N.S.)* 63(4):793–816.

Axlon. 1982. Supercharge Your APPLE II (advertisement). *Creative Computing* 8(4):131.

Aycock, John. 2021. The Coming Tsunami of Digital Artefacts. *Antiquity* 95(384):1584–1589. https://doi.org/10.15184/aqy.2021.84.

Aycock, John. 2023. Amnesia *Remembered: Reverse Engineering a Digital Artifact*. Berghahn, New York.

Aycock, John, and Katie Biittner. 2019. Inspecting the Foundation of *Mystery House*. *Journal of Contemporary Archaeology* 6(2):183–205. https://doi.org/10.1558/jca.36745.

Aycock, John, and Katie Biittner. 2020. Le*GACy* Code: Studying How (Amateur) Game Developers Used Graphic Adventure Creator. *Proceedings of the 15th International Conference on the Foundations of Digital Games* 23:1–7. https://doi.org/10.1145/3402942.3402988.

Aycock, John, Shankar Ganesh, Katie Biittner, and Paul Allen Newell. 2022. The Sincerest Form of Flattery: Large-Scale Analysis of Code Re-Use in Atari 2600 Games. *Proceedings of the 17th International Conference on the Foundations of Digital Games* 26:1–10. https://doi.org/10.1145/3555858.3555948.

Bredon, Glen. 1984a. *Merlin: The Macro Assembler for the Apple*. Roger Wagner, Santee, California.

Bredon, Glen. 1984b. *Merlin Pro: The Macro Assembler for the Apple IIe & IIc*. Roger Wagner, Santee, California.

Bruhns, Karen Olsen. 1972. Two Prehispanic Cire Perdue Casting Moulds from Colombia. *Man (N.S.)* 7(2):308–311.

Carr, Philip J., and Andrew P. Bradbury. 2010. Flake Debris and Flintknapping Experimentation. In *Designing Experimental Research in Archaeology: Examining Technology through Production and Use*, edited by Jeffrey R. Ferguson, pp. 71–92. University Press of Colorado, Boulder.

Carver, Martin. 2009. *Archaeological Investigation*. Routledge, London.

Commodore Business Machines. 1982a. *The Commodore 64 Macro Assembler Development System*. Commodore Business Machines, West Chester, Pennsylvania.

Commodore Business Machines. 1982b. *Commodore 64 Programmer's Reference Guide*. Commodore Business Machines, Wayne, Pennsylvania.

Congdon, L. O. K. 1985. Water-Casting Concave-Convex Wax Models for Cire Perdue Bronze Mirrors. *American Journal of Archaeology* 89(3):511–515. https://doi.org/10.2307/504365.

Cook Inlet Tribal Council. 2017. Storytelling for the Next Generation: How a Nonprofit in Alaska Harnessed the Power of Video Games to Share and Celebrate Cultures. In *The Interactive Past: Archaeology, Heritage & Video Games*, edited by Angus A. A. Mol, Csilla E. Ariese, Krijn H. J. Boom and Aris Politopoulos, pp. 21–31. Sidestone Press, Leiden, Netherlands.

https://doi.org/10.1017/aap.2024.5 Published online by Cambridge University Press

Crabtree, Don. 1975. Comments on Lithic Technology and Experimental Archaeology. In *Lithic Technology: Making and Using Stone Tools*, edited by Earl Herbert Swanson, pp. 105–114. Mouton, The Hague, Netherlands.
Curwen, E. Cecil. 1930. Prehistoric Flint Sickles. *Antiquity* 4(14):179–186.
Curwen, E. Cecil. 1935. Agriculture and the Flint Sickle in Palestine. *Antiquity* 9(33):62–66.
Daragan, Marina N., and Yurij N. Romanenko. 2021. Technique and Technology of Scythian Bronze Arrowhead Casting: Experimental and Metallographic Approach. *Journal of Archaeological Science: Reports* 37:102919. https://doi.org/10.1016/j.jasrep.2021.102919.
David, Nicholas. 2017. The Antikythera Mechanism: Its Dating and Place in the History of Technology. *Journal of Mediterranean Archaeology* 30(1):85–104.
Goren, Yuval. 2008. The Location of Specialized Copper Production by the Lost Wax Technique in the Chalcolithic Southern Levant. *Geoarchaeology* 23(3):374–397.
Goren, Yuval. 2014. Gods, Caves, and Scholars. Chalcolithic Cult and Metallurgy in the Judean Desert. *Near Eastern Archaeology* 77(4):260–266.
Graham, Shawn. 2020a. An Approach to the Ethics of Archaeogaming. *Internet Archaeology* 55. https://doi.org/10.11141/ia.55.2.
Graham, Shawn. 2020b. *An Enchantment of Digital Archaeology: Raising the Dead with Agent-Based Models, Archaeogaming, and AI*. Berghahn, New York.
Hanussek, Benjamin. 2019. Conducting Archaeogaming and Protecting Digital Heritage: Does the Future for Archaeology Lie in The Immaterial. *Art and Science* 3(1). https://doi.org/10.21494/ISTE.OP.2019.0414.
Mathieu, James R. 2002. Introduction. In *Experimental Archaeology: Replicating Past Objects, Behaviors, and Processes*, edited by James R. Mathieu, pp. 1–11. BAR International Series 1035. Archaeopress, Oxford.
Moshenska, Gabriel. 2016. Reverse Engineering and the Archaeology of the Modern World. *Forum Kritische Archäologie* 5:16–28.
MOS Technology. 1976. *MCS6500 Microcomputer Family Hardware Manual (January 1976)*. MOS Technology, Norristown, Pennsylvania.
Newell, Paul Allen, John Aycock, and Katie Biittner. 2022. Still *Entombed* after All These Years: The Continuing Twists and Turns of a Maze Game. *Internet Archaeology* 59. https://doi.org/10.11141/ia.59.3.
Outram, Alan K. 2008. Introduction to Experimental Archaeology. *World Archaeology* 40(1):1–6. https://doi.org/10.1080/00438240801889456.
Pape, Bob. 2013. *It's Behind You: The Making of a Computer Game*. Electronic document, https://www.bizzley.com, accessed November 21, 2023.
Parallax. 2021. Our Story. Electronic document, https://www.parallax.com/our-story/, accessed August 2, 2023.
Pilgrim, Mark. 2014. Emulator Detection in 6502 Assembly (Kansasfest 2014 Presentation). Electronic document, https://archive.org/details/EmulatorDetectionIn6502AssemblyKansasfest2014Presentation, accessed November 22, 2023.
Politopoulos, Aris, Angus A. A. Mol, and Sybille Lammes. 2023. Finding the Fun: Towards a Playful Archaeology. *Archaeological Dialogues* 30(1):1–15.
Rassalle, Tine. 2021. Archaeogaming: When Archaeology and Video Games Come Together. *Near Eastern Archaeology* 84(1):4–11.
Reinhard, Andrew. 2013. What Is Archaeogaming? Electronic document, https://archaeogaming.com/2013/06/09/what-is-archaeogaming/, accessed August 25, 2023.
Reinhard, Andrew. 2015. Excavating Atari: Where the Media was the Archaeology. *Journal of Contemporary Archaeology* 2(1):86–93.
Reinhard, Andrew. 2018. *Archaeogaming: An Introduction to Archaeology In and Of Video Games*. Berghahn, New York.
Reinhard, Andrew. 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):855–884.
Rice, Michael. 1994. *The Archaeology of the Arabian Gulf, c. 5000–323 BC*, Routledge, London.
Rose, Thomas, Peter Fabian, and Yuval Goren. 2023. The (In)visibility of Lost Wax Casting Moulds in the Archaeological Record: Observations from an Archaeological Experiment. *Archaeological and Anthropological Sciences* 15:31. https://doi.org/10.1007/s12520-023-01731-6.
Salvador, Phil. 2023. Survey of the Video Game Reissue Market in the United States. Video Game History Foundation and Software Preservation Network. Electronic document, https://doi.org/10.5281/zenodo.8161056, accessed April 8, 2024.
Schick, Kathy D., and Nicholas Toth. 1994. *Making Silent Stones Speak: Human Evolution and the Dawn of Technology*. Simon and Schuster, New York.
Sedig, Jakob W. 2019. Ancient DNA's Impact on Archaeology: What Has Been Learned and How to Build Strong Relationships. *SAA Archaeological Record* 19(1):26–32.
Silas, Fred R. 2005. *Lost-Wax Casting: Old, New, and Inexpensive Methods*. Woodsmere Press, Pendleton, South Carolina.
Smith Nicholls, Florence. 2018. Archaeogaming as Queergaming. *Florence Smith Nicholls: Games and heritage* (blog), September 23. https://florencesmithnicholls.com/2018/09/23/archaeogaming-as-queergaming/, accessed August 25, 2023.
Smith Nicholls, Florence. 2021. Fork in the Road: Consuming and Producing Video Game Cartographies. In *Return to the Interactive Past: The Interplay of Video Games and Histories*, edited by Csilla E. Ariese, Krijn H. J. Boom, Bram van den Hout, Angus A. A. Mol, and Aris Politopoulos, pp. 117–133. Sidestone Press, Leiden, Netherlands.
Smith Nicholls, Florence, and Michael Cook. 2022. The Dark Souls of Archaeology: Recording Elden Ring. *Proceedings of the 17th International Conference of the Foundations of Digital Games* 17:1–10. https://doi.org/10.1145/3555858.3555889.
Spurrell, F. C. J. 1892. Notes on Early Sickles. *Archaeological Journal* 49:53–68.
Steil, Michael. 2011. How Many Commodore 64 Computers Were Really Sold? *Pagetable.com* (blog), February 1. https://www.pagetable.com/?p=547, accessed August 1, 2023.
Stout, Dietrich, Michael J. Rogers, Adrian V. Jaeggi, and Sileshi Semaw. 2019. Archaeology and the Origins of Human Cumulative Culture: A Case Study from the Earliest Oldowan at Gona, Ethiopia. *Current Anthropology* 60(3):309–340.
Taylor, Graham. 1984. And Pigs Will Fly . . . *Popular Computing Weekly* 3(14):12–13.
Titan Technologies. 1984. *Accelerator IIe Operations Manual*. Titan Technologies, Ann Arbor, Michigan.
Toth, Nicholas. 1985. The Oldowan Reassessed: A Close Look at Early Stone Artifacts. *Journal of Archaeological Science* 12(2):101–120.
Wang, Zhen, Lirun Yan, Ying Ma, Anding Shao, Jianjun Mei, and Kunlong Chen. 2023. Pilot Study on the Lipid Residues in the Clay Core of Lost-Wax Process. *Journal of Archaeological Sciences: Reports* 49. https://doi.org/10.1016/j.jasrep.2023.103990.
Winter, Matthew. 2021. Beyond Tomb and Relic: Anthropological and Pedagogical Approaches to Archaeogaming. *Near Eastern Archaeology* 84(1):12–21.
Wisseman, Sarah U., and Wendell S. Williams. 2013. Why Study Artifacts? An Interdisciplinary Approach. In *Ancient Technologies and Archaeological Materials*, edited by Sarah U. Wisseman and Wendell S. Williams, pp. 3–13. Routledge, London.

AUTHOR INFORMATION

John Aycock ■ Department of Computer Science, University of Calgary, Calgary, Alberta, Canada (aycock@ucalgary.ca, corresponding author)

Katie Biittner ■ Department of Anthropology, Economics & Political Science, MacEwan University, Edmonton, Alberta, Canada (biittnerk@macewan.ca)

https://doi.org/10.1017/aap.2024.5 Published online by Cambridge University Press