FrAG: A Framework for the Analysis of Games

Shankar Ganesh
Computer Science
University of Calgary
Calgary, Canada
sankarasubramanian.g@ucalgary.ca

John Aycock
Computer Science
University of Calgary
Calgary, Canada
aycock@ucalgary.ca

Katie Biittner
Anthropology, Economics & Political Science
MacEwan University
Edmonton, Canada
biittnerk@macewan.ca

***Abstract*—FrAG is a *Fr*amework for the *A*nalysis of *G*ames, allowing us to harness game-playing AI techniques to assist in the reverse engineering of historical video games for which the binary form of the game is the only thing remaining. Moreover, FrAG allows us to perform this task at scale, and search for patterns of game development that might otherwise be overlooked. Game development is done by humans, of course; our interdisciplinary approach is not exclusively technical, and also involves the field of archaeology and its well-established theories of humans and technology. Here we present FrAG's architecture along with our preliminary results using the framework on a test suite of Atari 2600 games.**

***Index Terms*—AI, reinforcement learning, binary reverse engineering, Atari 2600, archaeology, archaeogaming**

I. Introduction and Background

Many games exist from early in the history of the videogame industry for which we only have partial information. Frequently this means that we have a binary image of a game's code and data, but that the source code along with other records of development have been lost. Furthermore, the clever implementation techniques used by early game programmers for the highly constrained platforms of the day have become poorly known, thanks to our separation in time from that era. One way to regain some of this knowledge is by reverse engineering the binary forms of the games, a labor-intensive process, and indeed there are cases of individual games being reverse engineered (e.g., [1], [2]).

From a high-level point of view, reverse engineering techniques can be divided into static analysis and dynamic analysis [3]. Static analysis refers to examining the binary code and data without running the code; dynamic analysis, by contrast, occurs while the program being analyzed is running. While both approaches have their individual strengths, static analysis struggles in areas like deliberate code obfuscations [4] that may be hard to understand and hard to distinguish from code optimizations, and the undecidable problem of accurately separating code from data when the two are intermingled [5]. Dynamic analysis, meanwhile, can be expensive – the code must be run – but provides an unobstructed view of what code and data are used and how.

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

979-8-3503-2277-4/23/$31.00 ©2023 IEEE

Another important consideration is the question of scale: there are a lot of games. We are aware of only a few attempts to examine game corpora *en masse* in the literature, for which [6] uses exclusively static analysis; [7] is primarily based on static analysis but has the advantage of analyzing a simpler domain-specific interpreted language rather than machine code.

There has been extensive work recently using game-playing prowess as a test for reinforcement learning algorithms (e.g., [8]–[13]), much of which conveniently uses early Atari 2600 games. We take advantage of these advances to create FrAG, a *Fr*amework for the *A*nalysis of *G*ames, which marries a reinforcement learning algorithm with a game console emulator. Our insight is that, as a natural side effect of playing the games, these reinforcement learning algorithms – which we will colloquially refer to as AI – allow us to tap into a continuous stream of automatically generated data from a game's execution that facilitates dynamic reverse engineering.

It is important to note that we are using AI in a non-traditional way: we are *not* focusing on tuning the AI, and are effectively treating it as a black box. The focus for our application is not how well the AI can play the game, but on how completely a game's code and data are exercised. In actual fact, even failures to exercise certain code and data can reveal valuable information for reverse engineering, because they automatically identify areas in a game of interest for a human reverse engineer – this makes the best use of human analysis time and, for instance, might reveal unused content, or areas in a game where Easter eggs are present.

And, because we are only interested in running the game, we are collecting useful data even during the AI's training phase. We thus conserve resources and do not train the AI to a point where it might be considered "fully trained" in normal usage; the results in Section IV justify this approach. For most games, our data indicates diminishing returns when training beyond five million video frames. Before we dive into the technical details of FrAG, however, it is helpful to understand why we are doing this research and what its benefits are.

II. Why FrAG?

Videogames are not just technical objects appearing in isolation. They are the result of human creativity and labor, and reflect the culture in which they were developed and anticipated to be used. A field of study that has extensively examined the interactions between humans and technology is

archaeology, and our work with FrAG fits into the area of archaeology called *archaeogaming*.

Archaeogaming 'is the archaeology both in and of digital games' [14, p. 2]. It is an interdisciplinary field of research predominantly undertaken by archaeologists in collaboration with computer scientists, digital media researchers, and others engaged in understanding the games we build and play. It began by looking at how archaeological methods could be used in-game and at representations of archaeologists, archaeology, and archaeological sites [14] to a broader field of investigation, including how digital games can be used for cultural heritage management and for archaeological research dissemination. Our research within archaeogaming has focused on the implementation of video games through the analysis of code as artefact. From an archaeological point of view, artefacts are anything made, used, or modified by humans. Code certainly fits those criteria but defies the traditional understanding of the materiality of the objects/products of human technological labor and activities.

FrAG's ability to consider games at scale allows us to go beyond single artefacts, however, and study a whole corpus of games as *assemblages*. Traditionally assemblages are defined as a related set of things, a group of similar archaeological materials, from which information can be extracted because they are combinations of materials in context [15]. Context is a central tenant of archaeology; it is the spatial-temporal location of an object and provides the basis for interpretation. With studying digital artefacts, however, we challenge the materiality of this definition [16], and instead we treat the technological organization as context. An organization of technology approach 'helps us to conceive of the forms of technological agency invested in video games and their material manifestations' [17, p. 8].

Our own examination of technological organization uses the chaîne opératoire or operational sequence [18], [19]. It is a theoretical framework and conceptual model for the analysis of assemblages that 'seeks to reconstruct the organization of a technological system at a given archaeological site' [19, p. 106]. We have examined the organization of technology via the chaîne opératoire by reconstructing the sequence of decisions and operations that transform a raw material (or in the case of digital artefacts, code) into a product, a cultural commodity: a game. Thus a corpus of games is not just a grouping of things but is an assemblage that is a productive, dynamic process composed of events – an understanding we can use to interpret the relations and interactions between the code and the people building and using it.

III. FrAG Architecture

The emulator FrAG uses is version 0.252 of the open-source Multiple Arcade Machine Emulator (MAME).[1] While MAME is able to emulate more than 30,000 systems, we have focused on the Atari 2600 game console in this current work, although we do have a proof-of-concept implementation of FrAG with the Fairchild Channel F console that demonstrates our framework can be used with other devices; nothing we are aware of prevents applying this approach to other platforms.

The Atari 2600 we target is a video game console developed and first released by Atari in 1977 [20]. The console contained three chips of note: CPU, RAM-I/O-timer, and custom video/audio, with its games residing in ROM inside pluggable cartridges. The system was highly constrained, with a mere 128 bytes of RAM and directly-addressable ROM limited to at most 4 KiB. The system and its games are of historical importance, though, because as Montfort and Bogost put it, 'although it was not the first home videogame console, the Atari VCS [2600] was the first wildly popular one' [21, p. 4].

FrAG itself is implemented in C++. It does not provide an AI, but instead provides a platform for training and running AI agents; FrAG's AI interface is based on that of the Arcade Learning Environment [8], or ALE, permitting ALE and ALE-compatible work to be easily leveraged. FrAG acts as the conduit through which the AI can supply input to the game under analysis, and also feeds back RAM values and image frame data to the AI for its learning algorithm. As well, FrAG uses MAME's debugging and memory monitoring functionality to extract dynamic analysis data as the AI is running the game. A variety of Python scripts and other programs are then used to postprocess the data from FrAG.

Unless stated otherwise, the reinforcement learning method used for the results here is a Deep Q-Network (DQN – see [10], [22]) from Stable Baselines 3 [23] with hyperparameters set per [10], again stressing that we are treating the AI as a black box. Some minimal system- and game-specific information is required for the AI to function. For the former, all valid game controller actions and combinations must be specified; the latter includes where to find a game's score data and the number of player lives in memory, so the AI can determine if progress has been made. Gathering game-specific information requires a minor amount of manual reverse engineering work for each game prior to its analysis with FrAG.

IV. Preliminary Results

To gather some preliminary data on FrAG's efficacy, we selected a test suite of eight games. These games were chosen to exhibit different properties (e.g., ROM size, game controller type), and were ones for which we had established ground truth for their code and data, either through original source code or a complete human reverse-engineered disassembly.

Table I shows the amount of code and data usage FrAG identifies when training DQN for different numbers of video frames. CPU instructions may occupy between 1–3 bytes each, and ROM sizes can exceed the directly addressable 4 KiB mentioned above because some games' cartridges would contain extra circuitry to switch between different ROM banks. While the data collected by FrAG are accurate, there is some slight ambiguity in how data is classified for two reasons. First, programmers would sometimes deliberately overlap code with data and even code with *other* code; we interpret the former as code and the latter as data. Second, the CPU will fetch

[1] https://www.mamedev.org/

additional bytes for some instructions that are never used, which given certain execution patterns and juxtapositions of code can erroneously look like code/data overlaps, but those cases are rare.

Overall the coverage by FrAG with DQN is very good, with code and data coverage 85% and higher on all but two cases. For *Combat*, where the coverage is lower, the gap is in the data, because over 90% of the code instructions are seen. The AI only plays 1 of a possible 27 game variants, and code for the second player is never exercised as multi-player capability is supported by FrAG but currently not the AI algorithm we used. Meanwhile, the results for *Montezuma's Revenge* are unsurprising because the (lack of) useful training feedback from the game makes it especially challenging [24], but here FrAG's ability to exchange DQN for other learning methods permits better techniques to be used as needed – we have substituted in Actor-Critic [25] and PPO learning methods [26] in other experiments, and Agent57, for instance, reports excellent gameplay results for *Montezuma's Revenge* [11].

Through FrAG's results we also begin to see potential programming practices exposed. Developers, pressed for ROM space, would use a conditional branch instruction as an unconditional branch to save a byte when they knew that the condition would always hold. Because FrAG's data can keep track of branch instruction behavior, we can use that data to definitely rule out the practice in many cases, allowing a reverse engineer to focus their time on the remaining branches. Table II shows this data for the games in the test suite.

Finally, FrAG's execution of a game provides data about dynamic execution that can inform reverse engineering. Figure 1 shows heatmaps of RAM usage for *Boxing*, for example, where we see the memory from 0xa4–0xab read extremely frequently, with write frequency lower, and focused on every other location. For reverse engineering, this suggests that those memory locations may hold (16-bit little-endian) pointers for data shown onscreen, where only the pointers' least-significant byte needs updating. And, while space precludes its inclusion, FrAG also gathers data about "hot" code that is frequently executed in addition to automatically detecting the screen-update ("kernel") code in a game. As future work, the data gathering could potentially be adapted to game mapping [27] and game design understanding [28] seen in some related research.

TABLE I
ROM COVERAGE FOR INCREASING DQN TRAINING

	Code (instructions)	Data (bytes)	Unused (bytes)	Total used (% bytes)
2 KiB ROM				
Boxing				
Ground truth	747	647	3	99.9
FrAG: 5 M frames	738	644	20	99.0
FrAG: 10 M frames	738	644	20	99.0
FrAG: 15 M frames	738	644	20	99.0
Combat				
Ground truth	770	571	8	99.6
FrAG: 5 M frames	696	232	490	76.1
FrAG: 10 M frames	696	228	494	75.9
FrAG: 15 M frames	696	232	490	76.1
Kaboom!				
Ground truth	847	428	0	100.0
FrAG: 5 M frames	815	419	70	96.6
FrAG: 10 M frames	815	420	69	96.6
FrAG: 15 M frames	815	420	69	96.6
4 KiB ROM				
Dragonfire				
Ground truth	1574	1158	8	99.8
FrAG: 5 M frames	1437	884	526	87.2
FrAG: 10 M frames	1446	911	483	88.2
FrAG: 15 M frames	1446	922	472	88.5
River Raid				
Ground truth	1588	1066	1	100.0
FrAG: 5 M frames	1517	851	351	91.4
FrAG: 10 M frames	1520	863	334	91.8
FrAG: 15 M frames	1521	863	332	91.9
8 KiB ROM				
Montezuma's Revenge				
Ground truth	2914	2388	2	100.0
FrAG: 5 M frames	2103	1278	2734	66.6
FrAG: 10 M frames	2341	1398	2145	73.8
FrAG: 15 M frames	2343	1408	2130	74.0
Moonsweeper				
Ground truth	2890	2752	0	100.0
FrAG: 5 M frames	2546	2238	1174	85.7
FrAG: 10 M frames	2587	2241	1097	86.6
FrAG: 15 M frames	2587	2243	1095	86.6
16 KiB ROM				
Solaris				
Ground truth	4957	6694	0	100.0
FrAG: 5 M frames	4905	6149	651	96.0
FrAG: 10 M frames	4897	6145	671	95.9
FrAG: 15 M frames	4904	6167	636	96.1

V. CONCLUSION

By taking advantage of improvements in AI game-playing, we use the side effects of AI-driven gameplay to gather dynamic execution data within FrAG, our framework for the analysis of games. While initially employed with Atari 2600 games, the framework can be retargeted to other game consoles and have its game-playing AI algorithm changed as necessary to embrace further research advances.

Our preliminary results show that even without making heroic efforts at tuning, we obtain good coverage of code and data for most games in our test suite, and are able to gather data to assist reverse engineering and uncover programming practices. Ultimately, FrAG is a framework that will help us explore, analyze, and understand the human activity of game development at scale through the lens of archaeology.

TABLE II
CONDITIONAL BRANCH INSTRUCTION BEHAVIOR

Game	Conditional branches	Conditional branches always taken
Boxing	108	9
Combat	89	23
Dragonfire	233	39
Kaboom!	129	15
Montezuma's Revenge	414	90
Moonsweeper	429	58
River Raid	278	56
Solaris	618	22

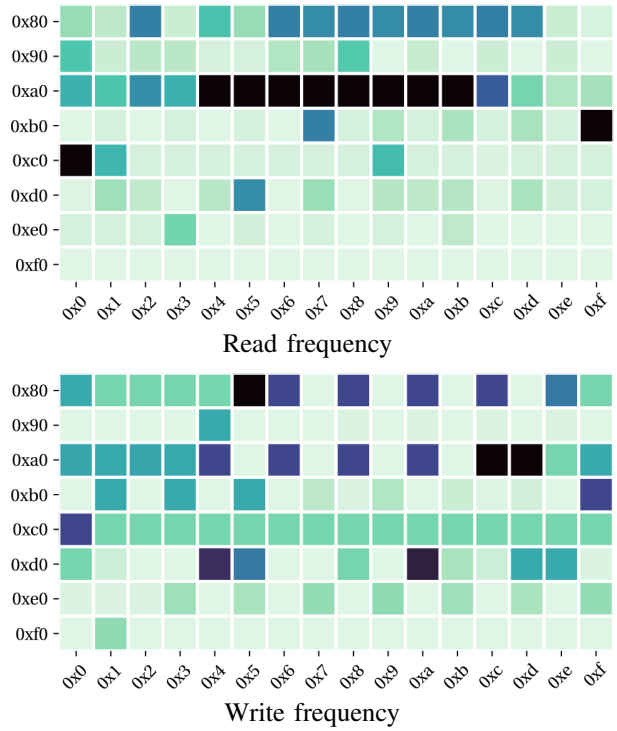

Fig. 1. RAM usage heatmaps for *Boxing*.

ACKNOWLEDGMENT

Thanks to Ana Bindiu, Rachel Ralph, and Ella Tomlinson for gathering data to run games in FrAG, and to Matthew Michaud for adapting it to work with the Fairchild Channel F.

REFERENCES

[1] J. Aycock and K. Biittner, "Inspecting the foundation of *Mystery House*," *Journal of Contemporary Archaeology*, vol. 19, no. 2, pp. 183–205, 2019.

[2] L. Wiest, "Reverse engineering Star Raiders," *PoC||GTFO*, vol. 0x13, pp. 5–20, 2016.

[3] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Computing Surveys*, vol. 44, no. 2, 2012.

[4] C. Collberg and J. Nagra, *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection*. Addison-Wesley, 2010.

[5] R. N. Horspool and N. Marovac, "An approach to the problem of detranslation of computer programs," *The Computer Journal*, vol. 23, no. 3, pp. 223–229, 1980.

[6] J. Aycock, S. Ganesh, K. Biittner, P. A. Newell, and C. Therrien, "The sincerest form of flattery: Large-scale analysis of code re-use in Atari 2600 games," in *17th International Conference on the Foundations of Digital Games*, 2022.

[7] J. Aycock and K. Biittner, "LeGACy code: Studying how (amateur) game developers used Graphic Adventure Creator," in *15th International Conference on the Foundations of Digital Games*, 2020.

[8] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The Arcade Learning Environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.

[9] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling, "Revisiting the Arcade Learning Environment: Evaluation protocols and open problems for general agents," *Journal of Artificial Intelligence Research*, vol. 61, pp. 523–562, 2018.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 02 2015.

[11] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, D. Guo, and C. Blundell, "Agent57: Outperforming the Atari human benchmark," Mar. 2020, arXiv:2003.13350 [cs.LG].

[12] V. Volz and B. Naujoks, "Towards game-playing AI benchmarks via performance reporting standards," in *IEEE Conference on Games*, 2020, pp. 764–771.

[13] S. Kapturowski, V. Campos, R. Jiang, N. Rakićević, H. van Hasselt, C. Blundell, and A. P. Badia, "Human-level Atari 200x faster," Sep. 2022, arXiv:2209.07550 [cs.LG].

[14] A. Reinhard, *Archaeogaming: An Introduction to the Archaeology in and of Video Games*. Berghahn, 2018.

[15] T. D. Price and K. J. Knudson, *Principles of Archaeology*, 2nd ed. Thames and Hudson, 2018.

[16] T. Ingold, "Materials against materiality," *Archaeological Dialogues*, vol. 14, no. 1, pp. 1–16, 2007.

[17] O. Sotamaa, "Artifact," in *Routledge Companion to Video Game Studies*, M. J. P. Wolf and B. Perron, Eds. Taylor and Francis/Routledge, 2014, pp. 3–9.

[18] P. Lemonnier, "The study of material culture today: Toward an anthropology of technical systems," *Journal of Anthropological Archaeology*, vol. 5, no. 2, pp. 147–186, 1986.

[19] F. Sellet, "Chaîne opératoire: The concept and its applications," *Lithic Technology*, vol. 18, no. 1/2, pp. 106–112, 1993.

[20] J. Decuir, "Atari Video Computer System: Bring entertainment stories home," *IEEE Consumer Electronics Magazine*, vol. 4, no. 3, pp. 60–66, 2015.

[21] N. Montfort and I. Bogost, *Racing the Beam: The Atari video computer system*. MIT Press, 2009.

[22] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," Dec. 2013, arXiv:1312.5602 [cs.LG].

[23] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[24] A. Garriga-Alonso, "Solving Montezuma's Revenge with planning and reinforcement learning," BSc thesis, Universitat Pompeu Fabra, 2016. [Online]. Available: http://hdl.handle.net/10230/30867

[25] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," Jun. 2016, arXiv:1602.01783 [cs.LG].

[26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," Aug. 2017, arXiv:1707.06347 [cs.LG].

[27] J. C. Osborn, A. Summerville, N. Dailey, and S. Lim, "MappyLand: Fast, accurate mapping for console games," in *17th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2021, pp. 66–73.

[28] J. C. Osborn, A. Summerville, and M. Mateas, "Automated game design learning," in *IEEE Conerence on Computational Intelligence and Games*, 2017, pp. 240–247.