# Game and Interactive Software Scholarship Toolkit (GISST)

Eric Kaltman, Joseph Osborn, Noah Wardrip-Fruin, Michael Mateas
Expressive Intelligence Studio
UC Santa Cruz
Santa Cruz, California 43017-6221
ekaltman@soe.ucsc.edu

## ABSTRACT

The Game and Interactive Software Scholarship Toolkit (GISST) is a suite of tools aimed at helping support and create new forms of game studies and software studies scholarship. Digital games scholars generally analyze game objects using ad-hoc methods: some games are emulated and others are recorded as video or screenshots, and scholars rarely explicitly define their methodologies for working with game software (or performances employing that software) or reference games in standardized ways. GISST supports game scholars by providing for the creation, reference and management of games (as executable data), game performances (as video) and game states (as frozen run-time memory from a suite of supported emulators). GISST can ingest a variety of game studies-related resources and reproduce them as Web documents through its `CiteState.js` JavaScript library. GISST is a first step in a new methodology focused on the creation of support applications for work in game studies, software studies and game history.

## KEYWORDS

citation, game studies, preservation, performance, emulation

**ACM Reference format:**
Eric Kaltman, Joseph Osborn, Noah Wardrip-Fruin, Michael Mateas. 2017. Game and Interactive Software Scholarship Toolkit (GISST). In *Proceedings of International Conference on the Foundations of Digital Games, Cape Cod, MA, USA, August 2017 (FDG '17),* 4 pages.
DOI:

## 1 INTRODUCTION

In working with digital games, researchers confront an array of technical and resource management issues that get in the way of work. Older games require older hardware, or the re-articulation of their data through emulation. To analyze or display parts of games in arguments, scholars need to use a range of tools for screen capture and video recording. These issues are compounded by the restrictions of current publication platforms and lack of standard practices. How does one reference a game in a precise and stable way? Where does one store videos of gameplay performance? How does one reference and retrieve an emulated ROM? What would be possible for games scholarship if there were an archival-quality resource for the management of digital games and the secondary artifacts incumbent to their study? The Game and Interactive Software Scholarship Toolkit (GISST) is a suite of tools which facilitates the organization, reference, analysis, and retrieval of games, game performances (video), and game executable states (emulated save states), and moreover provides for their inclusion in digital publications. This is a first step towards better platforms for the citation and retrieval of games and their related documentation.


*FDG '17, Cape Cod, MA, USA*

DOI:

## 2 THE GAME AND INTERACTIVE SOFTWARE TOOLKIT

GISST works with three classes of objects: games, performances, and executable game states.[1] "Games" in this sense are collections of data about a game concept. This includes both basic descriptive information — required for correct bibliographic entries — and the executable data needed to run a game. In GISST's case, executable data can be run in a suite of browser-based emulators. Performance objects are records of games as played or viewed by a player or group. These records also carry both descriptive metadata and the viewable performance data that metadata describes. "Viewable performance data" is either a collection of frames — GIFs or video — representing some situated act of play, or replay data — input stream recordings for emulators or replay files for a specific game engine. Executable game states are snapshots of a game's run time memory, either saved by an emulator into a "save state" file, or extracted directly from a system during execution. GISST manages a database of game, performance, and game state records and can embed executable game data or viewable performance data into Web pages.

GISST has three primary components:

(1) A command line interface (CLI) for ingesting game and performance data, generating metadata records, and managing the reference database.
(2) `CiteState.js`, a JavaScript emulation interface that acts as a wrapper for a suite of cross-compiled C-to-JavaScript emulators.
(3) A web application (the "app") that allows for basic search and record viewing, along with an "Indexer" for creating emulated game states, screenshots, and game play performance videos.

The three components are inter-related. The CLI's `serve` command launches the app and the CLI's reference store provides stable links for data displayed and used in the app. `CiteState.js` is a major component of the app's Indexer, which is basically a user interface for `CiteState.js`'s API described below. Figure 1 illustrates the relationships between the components. Input resources

[1] The demo is available as an installable Python package and as a pre-configured virtual machine at: https://github.com/gamecip/gisst

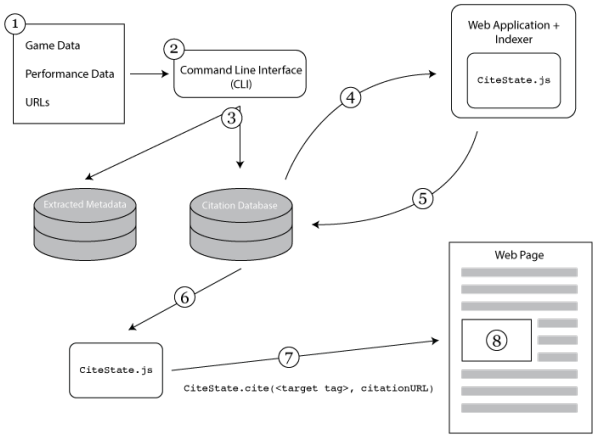


**Figure 1: GISST components and pipeline.**

(1) are fed to the CLI (2) which extracts their information (3) into an extraction table (for URLs) or the citation database (for performance and game data). The Web Application reads from the citation database (4) and the Indexer uses `CiteState.js` to create further citable resources (5). `CiteState.js` can then use those resources' permanent URLs (6) and its `cite` function (7) to embed an executable program into a target HTML tag (8).

## 2.1 GISST Command Line Interface

GISST's CLI is a Python script that ingests various citable resources, maps them to a citation schema, and places them into a citation store. There are two main commands for the CLI, `cite_game` and `cite_performance`. `cite_game` draws game data and metadata from a variety of different resource types. The `--file_path` and `--directory` flags ingest supported game ROM files or directories imaged from physical media, respectively. `--directory` will automatically search the folder for the first executable file it finds. If this is not the game's executable (many DOS games require installation programs to be run first), there is an `--executable` flag that lets the user provide the executable path. The `--url` flag links the record to information from common game informational databases (currently, Wikipedia or MobyGames). The scraped information can be linked to an ingested game resource to automatically flesh out the information in the citation store (a separate extraction store is used to capture additional metadata that is outside of general citation needs). Furthermore, thanks to community driven data archiving, tools already exist for validating some game file data. The CLI makes use of the UCon64 tool which provides checksum validation for every known game for the Nintendo Entertainment System (NES), Super Nintendo Entertainment System (SNES), and Nintendo 64 (N64). `cite_game`'s `--title` flag automatically conducts a title search on the MobyGames database and presents the user with an itemized, pre-generated listing of potential citations.

The `cite_performance` command uses an analogous `--file_path` flag to ingest performance videos and replay data for supported emulators. `cite_performance`'s `--url` flag takes YouTube video links as inputs and maps them into the citation store. This is useful for cases in which one wants to link the game played in an online video to its game's citation. The CLI's supported resource types are listed in Table 1.[2]

| Citation Type | Supported File Types | Supported URI Source |
|---|---|---|
| Game | .NES ROM format<br>.SMC ROM format<br>Directory with a DOS compiled executable<br>.z64 ROM Format | MobyGames<br>Wikipedia |
| Performance | FM2 Replay format<br>Generic Video Files | YouTube |

**Table 1: CLI supported file types.**

## 2.2 CiteState.js

`CiteState.js` is a JavaScript library which provides a *front-end* to any conforming JavaScript *runner program*. In theory, a runner could be any JavaScript code that renders audio and video via the *Web Audio* and *Canvas* APIs respectively and which only adds input event handlers to a specific HTML element. We currently assume (without loss of generality) that runners come from a fixed but extensible set of Emscripten-compiled emulators [6].[3] Beyond the constraints on input and output, we also require that runners can be initialized with URLs pointing to specific game binary and saved state resources, and that multiple instances can be run simultaneously. Emulators may also provide information on their emulated systems' memory regions and their sizes along with ways to read data from those regions.

With a library of runners which satisfy these API requirements, `CiteState.js` provides a unified interface to running, pausing, saving, loading, and recording games on a variety of platforms given only URLs obtained from, e.g., the reference database. Existing ports of emulation software to JavaScript via Emscripten required modification to comport with `CiteState.js` interface: for example, a port might not expose its rendering canvas in an easily accessible way, or might be set up to load arbitrary games and save states rather than a specific file reference. This was the case for existing ports of the NES (FCEUX), SNES (SNES9x) and N64 (Mupen64Plus) emulators [1, 3, 5]. For these ports, we altered their command-line drivers' arguments to look for games and save states in specific virtual filesystem locations, then wrote small JavaScript shims that ensured the correct files were loaded at runtime from specific URLs.

The DOS emulator (a port of DOSBox) posed a bigger challenge. DOSBox does not support saving or loading system states; moreover both the emulator and the underlying emulated system have their own virtual filesystems (defined by Emscripten and by DOSBox itself, respectively). We resolved the issue of saving and loading states as follows: since the Emscripten runtime is made of JavaScript objects and byte arrays, we save a state by copying out the byte arrays representing the heap along with the interpreter's bytecode stack

[2] FM2 replay files and .z64 ROMs are ingestible but not fully supported in the app.
[3] Emscripten is a Mozilla Foundation supported compiler that allows for the compilation of C programs into JavaScript.

(unlike the other emulators, DOSBox requires an interpreter as well as the regular Emscripten runtime). Loading comprises restoring that heap and stack and resetting the system clock appropriately. This approach would work for any Emscripten-based runner that does not support savestates — meaning that any C program compilable by Emscripten could potentially make use of the technique, not just emulators.

There was, however, an additional wrinkle: the filesystem. Because a DOS game might write to the emulated filesystem, and this exists outside of the running program's memory, therefore GISST copies out the filesystem state along with saved game states. This added addition overhead, requiring compression and tracking of files linked to each saved game state.

Because `CiteState.js` puts strong requirements on the runners' input and output modalities, it can seamlessly *record* gameplay video and timed input sequences. Full recordings of audio and video are large and unwieldy, so `CiteState.js` includes an MP4 encoder (an Emscripten build of `libav`, a fork of FFMPEG). This encoding is done in a separate worker thread to maintain interactive play.

## 2.3 GISST Web Application

The GISST Web Application provides for the in-browser management of the citation database, as well as the creation of new game performance videos, GIF video segments, and emulator save states. The web application consists of a Python backend linking to the citation store created by the CLI and a JavaScript UI application. The UI application provides for full-text search of stored citations, a full listing of the citation store's current contents, individual pages for each stored citation, and a wrapper tool for `CiteState.js` called the Indexer. As described in the last section, `CiteState.js` provides an API for generating various performance and internal resources for supported games. However, `CiteState.js` does not provide persistent storage or management of the data it produces. The Indexer embeds `CiteState.js` into a UI frame with user-friendly controls for `CiteState.js`'s various API functions. As shown in Figure 2, the Indexer provides both a single view of a game in the citation database and a comparative, simultaneous view of up to six concurrent emulated games. Users can save a state at any time, record video, and reload previously recorded performances and states.

Indexer takes the outputs of `CiteState.js`'s API functions, compresses them (in the case of video and DOSBox filesystem states), and uploads them to a server spawned by the CLI's `serve` command. Once uploaded, the video or save state data is stored in the citation database and made available as static links for `CiteState.js`. The Indexer automatically handles the creation of links between recorded game performances and save states.

Each video is automatically bookended with state saves to allow future users to continue (or create alternatives to) previous performances. If any states are saved during recording, the Indexer will automatically link those save states to their timestamps in the performance video. This allows a future user to jump into the performance recording at particularly salient points saved by an earlier researcher. Additionally if, while recording a performance, a new save state is loaded, the current performance is ended and a new, derived performance is started. Chains of derivation can be used to reveal alternate pathways through a game, or to show a variety of comparative gameplay actions at a specifically indexed point.

# 3 USE CASES

GISST's tools provide prototypical support for numerous common game analysis tasks that previously required ad-hoc and usually undocumented approaches. The system operates on explicit records of each game, performance, and game state, linking them to each other and tracking new derivative records created by users. It also provides the potential for shareable, stable links to emulated games that obviate the need for individual scholars to manage the technical complexity of running games in emulation and ensuring that future readers as always looking at the same articulation of an emulated game's data. This section highlights some basic use cases for GISST and how they support new analytical territory for game studies. For more extensive use case discussion please refer to [4].

## 3.1 Reference Creation and Sharing

One use for GISST is in the analysis of historical game titles supported by `CiteState.js`. For instance, one could extract the ROM data from a NES cartridge, say *Super Mario Bros.*, or download it from an enthusiast website (or in a hypothetical future, from a digital repository of game studies resources). In order to reference a specific location in the game, like "World 1-1", a scholar can use the CLI to run a `cite_game` command providing the ROM file as input. For example:

```
gisst cite_game --file_path super_mario.nes
```

The CLI uses the UCon64 ROM validation program to initially label the ROM data, and then allows the scholar to modify the record with additional information. To run the game, the user calls the CLI's `serve` command to start the app, which can then be loaded in a local web browser. From there, clicking on the game's listing will load it into the Indexer. The "Start Emulation" button will automatically load up the game in an in-browser window. The scholar can then start the game and then press the "Start Recording" button to begin a video recording of "World 1-1". Upon starting the recording, the Indexer will call `CiteState.js`'s `saveState` function, which returns a save state file generated by the underlying FCEUX emulation instance. The Indexer will link this file to both the game's record — created by the `cite_game` function — and as the beginning state of the current video recording. When finished with the level, the scholar can "Stop Recording," which will again let the Indexer save and link the closing state of the recording. In this example, the system has indexed and created explicit references to the *Super Mario Bros.* game data (ROM file), two saved game states (FCEUX save state files), and a video recording of "World 1-1" (as an MPEG-4 video file), all without the need to run more than two command line functions. We are currently developing ways to ingest these files through the browser interface, reducing the need for external CLI steps.

Since each referenced resource is stored at a stable, locally web-accessible URL, each item created in the last paragraph can be inserted directly into a standard web-page by including the `CiteState.js` library and writing a one-line JavaScript function — (7) in Figure 1.

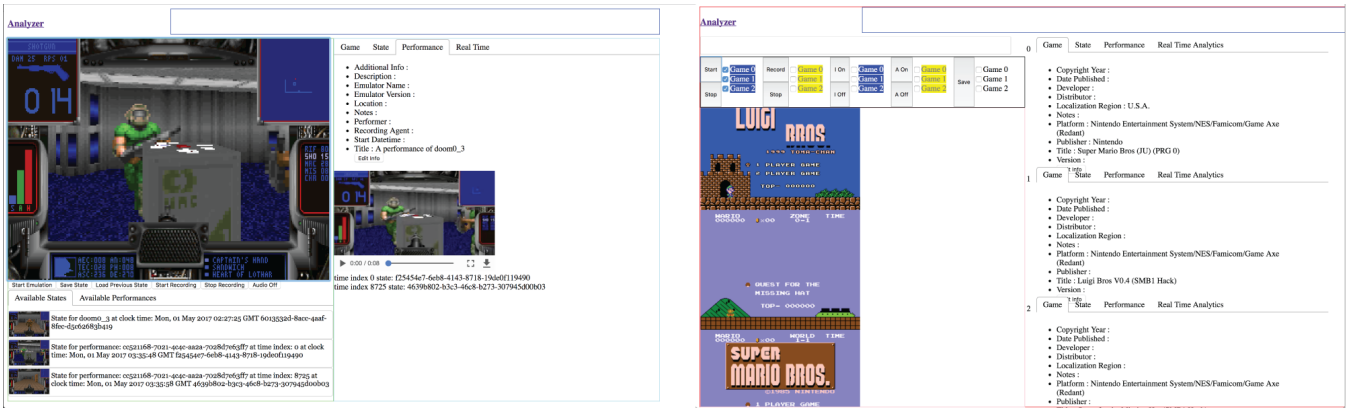


**Figure 2: Two versions of the Indexer UI. Left: Single game Indexer with program window, saved state listing, and view of previously recorded performance. Right: Comparative view with multiple selectors for input streams and sound.**

## 3.2 Unified Game Reference Database

GISST's division between the app and CLI-managed server is intended to prototype a future in which a separate, verified database of GISST references can exist and be shared amongst researchers. One scholar working on a specific title could search and recall specific in-game locations or videos of gameplay segments and then embed them in their work with assurance of its recovery by readers. Present game citation practices are underdeveloped, and it is hoped that GISST will prod further discussion of the preferred means for sharing and citing game references.

## 3.3 Comparative Analysis

As shown on the right side of Figure 2 the Indexer supports multiple, concurrent emulations running side-by-side. This enables analysis of game versions, sequels, localizations, and other potential comparative phenomena. The Indexer controls allow for the same functionality for each emulated instance — save, load, and video recording — with an additional provision for directing input streams to subsets of the emulations. In the emulator windows of the figure, a scholar compares multiple modified versions of *Super Mario Bros.* with the original title in the top left position. (This example is drawn from Shane Denson's work in digital seriality [2] and shows how GISST can fit into existing game studies workflows.)

# 4 FUTURE WORK

GISST is a fully-functional prototype. It allows knowledgeable users to explore the potential for tools of this type. But ideal future work would include making the software robust enough and the interface accessible enough for widespread use by game and software studies scholars. It would also include the creation of communities for use and upkeep of the software and its products.

Tools such as GISST open up numerous possibilities for the dissemination and standardization of references to game studies resources. For bibliographic references, all the information in a reference record could be exported to BibTeX or linked with scholarly citation systems like Zotero. Executable references have information about emulated execution embedded within their metadata, allowing for more consistent analytical use of emulation in game studies and potential future comparison of not only different games under emulation, but also different emulators themselves.

Since the in-browser emulation is essentially a full computing system running in a web page, its memory and operations are totally available to introspection via other concurrent JavaScript processes. Work is underway to include memory inspection and manipulation functionality in the `CiteState.js` API, allowing for future in-browser visualization of emulated games' low-level runtime behavior.

GISST's focus on referential structures derives from its initial focus on game citation management. Attempting to find a way to cite games, performances, and game states, incidentally required creating the technical means for those references, and figuring out how those references could be made available and useful for argumentation. This forced the conceptualization of new scholarly expressions, and brought about the realization that the citation apparatus was just the first step in the creation of a new class of applications aimed at the needs of games and software studies scholars. The components of GISST described in this abstract are then a preface to a whole range of possible tools for game history, game studies and software studies works.

# 5 ACKNOWLEDGMENTS

This work was made possible, in part, by Institute of Museum and Library Services grant LG-06-13-0205-13.

# REFERENCES

[1] 2017. Mupen64-Plus. (2017). https://github.com/mupen64plus
[2] Shane Denson. 2015. Digital Seriality. http://shanedenson.com/stuff/visualizing_digital_seriality/digital-seriality.html. (2015).
[3] Valtteri Heikkil a. 2016. EM-FCEUX. (2016). https://bitbucket.org/tsone/em-fceux
[4] Eric Kaltman, Joseph Osborn, Noah Wardrip-Fruin, and Michael Mateas. 2017. Getting the GISST: A Toolkit for the Creation, Analysis and Reference of Game Studies Resources. In *Proceedings of the 12th International Conference on the Foundations of Digital Games*. Hyannis, MA.
[5] tjwei. 2015. xnes. (2015). https://github.com/tjwei/xnes
[6] Alon Zakai. 2011. Emscripten: an LLVM-to-JavaScript compiler. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*. ACM, 301–312.