An Architecture for Repeatable, Large-Scale Educational Game Data Analysis: Building on Open Game Data

Luke Swanson(✉) and David J. Gagnon

University of Wisconsin-Madison, Madison, WI 53706, USA
lwswanson2@wisc.edu, djgagon@wisc.edu

Abstract. Given the incredible popularity of video games in contexts from entertainment to education, and the capacity of internet-connected games to record fine-grained telemetry data, there exists an unprecedented opportunity to investigate gameplay behaviors, outcomes, and their relationships to learning processes. However, with these opportunities come the need for technical infrastructures to manage the collection and analysis of massive amounts of game event data. In this work, we build upon existing literature to develop an architectural design for such infrastructure. We address issues of play data collection across many games; regular, repeatable extraction of gameplay features from raw data; and access to data for secondary analyses. In addition, we describe an implementation of this infrastructure and provide real-world examples of the implementation's usage in prior large-scale analysis work.

Keywords: Software architectures · Interactive learning environments · Computer games · Open source software

1 Introduction

Unfortunately, but not unexpectedly, collection and analysis of game data requires a significant amount of infrastructure and a diversity of expertise. A complete games-based research project must cover game design and development, distribution, data collection, data storage, data processing, visualization, analysis, and communication of research results. Thus, there are many potential barriers for new researchers or small teams to participate in game data research.

These barriers are exacerbated by the often closed nature of educational research. Closed science refers to a process of scientific inquiry where research questions, data, findings and results are kept confidential until publication, which still may limit access to subscribers or purchasers. This approach may lead to delayed dissemination, inaccuracy, lack of transparency and reproducibility, and inequality [11,17]. Open science attempts to confront these limitations, offering enhanced accessibility and equity [30], improved reproducibility and transparency [16], and better facilitation of collaboration [15]. The barriers to entering games research may be lowered by reducing the breadth of expertise required for each project. If game studios can simply be game studios, and researchers can simply be researchers, with shared infrastructure ensuring each group has the resources they need, it becomes easier for newcomers to participate in their area of expertise.

To this end, Open Game Data has been developed as a technical and social infrastructure for game data logging and analysis, which is used and developed by a growing number of educational video game researchers [4]. It is designed to support the general needs for data collection and processing across all kinds of video game, as well as the unique forms of learning data created by game environments. This infrastructure provides modular components for data collection, processing, access and analysis that support open science practices. Most importantly, it enables an open exchange of data between the studios that develop learning games, and the researchers that study them. As of the time of writing, the system processes 5–10M events daily from 17 games that use the system for analysis and distribution of public data sets.

However, there are issues that face such an initiative, and the simple pipeline described in past work on the subject. As the collection of games using shared infrastructure grows, there is a greater need for standards to ensure compatibility across games and over time. A need also arises to support diverse data analysis tools and techniques. Across the entire pipeline, an increasing scale in games, gameplay sessions, and analyses increases the number of points of failure in the analysis pipeline and thus creates a need for adequate monitoring of system performance. In this work, we expand upon the general pipeline described in the prior work, presenting an architecture for logging and processing of telemetry data across many games, and an implementation thereof.

2 Background

The commercial video game industry has a long tradition of leveraging data and using analytics approaches. A collection of several dozen industry and academic case studies were published in a volume edited by Seif El-Nasr et al. [1] demonstrating the value of using player data for topics such as ensuring game quality, maximizing success, understanding player behavior and enhancing the quality of the player experience. In educational contexts, the descriptive capacity of event log data affords many theoretical and methodological approaches. In games, learner performances can be conceptualized as both descriptive and procedural, and game researchers have adopted theories such as embodiment [3], socio-cultural learning [25], and situated learning [7]. Similarly, game studies using log data adopt a wide range of quantitative and qualitative methods such as discourse analysis [26], epistemic network analysis [22], player clustering [24,28], educational data mining [20], replay coding [19], learning engineering [2], and evidence centered design [12].

While many of these data mining and analytics projects leveraged their own proprietary systems and infrastructures, there are several examples of commercially available game analytics systems that are available for any researcher or developer today. One of the most popular systems is Google Analytics (GA). GA was initially developed for website analytics, specifically to understand user demographics, user behaviour and user acquisition. Game Analytics is a game-specific platform that is also widely used, and provides similar functionality to GA. Both platforms provide client libraries to work with popular programming languages and game engines as well as web-based visualization and reporting tools. However, both platforms focus on increasing monetization rather than understanding deeper features of the players' thinking or learning.

Several education-focused projects have also been developed in recent years that enable large-scale analytics and data mining. The ADAGE (Assessment Data Aggregator for Game Environments) game research platform was designed to collect and analyze data from digital game environments. Its purpose was to facilitate the study of player interactions, learning outcomes, and behavior within educational games, providing valuable insights for game designers and educators [27]. Unfortunately, the platform was never widely adopted beyond its developers, and is no longer in operation.

Successful tools for large-scale learning analytics include DataShop, a repository for educational data initiated by Carnegie Mellon University [13], and MORF (Massive Open Online Courses Research Framework), a platform for analyzing and sharing data from Massive Open Online Courses (MOOCs) [6]. DataShop has yielded numerous datasets, analytical tools, and insights into learning processes, significantly contributing to the fields of educational data mining and learning analytics. MORF provides a unique structure based in the use of Docker containers, which are self-contained virtual machine images, designed to ensure complete reproducibility while allowing the use of any software or language.

3 Architecture

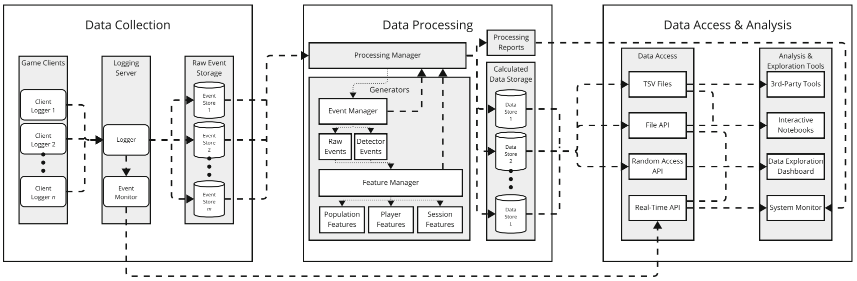

Fig. 1. Full architecture diagram, including all three blocks of the architectural design. Dashed lines indicate the primary flow of data from module to module. Dotted lines specifically indicate the flow of event data through the various generator modules.

With the projects discussed in 2 as a guide, we propose a modular infrastructure capable of serving large-scale data collection and analysis needs across many game projects. Our proposed system utilizes a layered architecture, allowing each layer can use different technical solutions at increasing scales of use, requiring only that the data exchanged between layers have a standardized format. There are three major layers, corresponding to primary phases of game data analysis. These are represented as top-level blocks within Fig. 1:

– Data Collection
– Data Processing
– Data Access and Analysis.

We describe each part of the architecture, and the relationships between them, in the following sections.

3.1 Data Collection

Data analysis necessarily begins with the collection of the data to be analyzed. In the case of game telemetry data, any number of players may interact with the game at any time. To enable consistent deployment of logging across many games and large numbers of players, we adopt a model using a single endpoint to route data from multiple game clients to one or more storage locations. This single entry point allows for the separation of event logging logic from the details of event storage. That is, changes in storage location or technology can be made without incurring maintenance costs to modify and re-deploy games that have not otherwise been changed. Only the entry point must be modified to redirect incoming data to a new storage endpoint.

Game Clients On the game/client side, a reusable logging package is imported by each game, rather than using ad-hoc implementations per-game. The package must adopt a standard event schema, such as the one proposed by Gagnon and Swanson [5]. Using a standard allows for the logging package to be ported to different game platforms without fear of diverging formats. Further, use of a single standard complements the single-point-of-entry system described in the "Logging Server" section; allowing differing data formats would necessarily increase the processing time needed to route data to storage, and may make the downstream portions of the data pipeline more difficult to manage. While the flexibility of using ad-hoc data formats for each game may offer some gains in initial development time, these are offset by the increased cost of long-term maintenance and support.

Logging Server As discussed above, our architecture uses a single server endpoint to collect all event logs sent by game clients. This creates a single entry point into the system for all clients across games and players, enabling monitoring of all data sources regardless of scale. The trade-off here is an emphasis on lightweight processing of incoming data, to avoid server overload. Thus, this server endpoint should act as a simple "router," identifying the source of an event, mapping it to a storage location, and generating an appropriate insertion request to the storage system. Optionally, this component of the architecture may be augmented with a load-balancing system, which reroutes incoming events to different logging server instances.

Real-Time Event Stream Monitoring One additional feature for the logging server is a mechanism for real-time monitoring of incoming events. The logging "router" mentioned above forwards each event to a monitor server, in addition to requesting insertion into the storage system. This monitor can then filter and/or process event data based on any active system monitor clients at the data access level (discussed in 3.3). While any server technology can potentially be used to implement such a monitor, we recommend a sockets-based server application, allowing for multiple concurrent connections to a single, persistent server instance.

Raw Event Storage We distinguish storage of raw event data from later, calculated events and data features. Data collected in this part of the architecture is treated as a series of objective, immutable events that occurred in the game. The logging server translates event data as directly as possible from the request sent by a logging client to the format of whatever storage technology is used. Once an event has been logged, it should only (at most) be moved to another location; event content is never modified or edited. Instead, post-processing may create and use "annotated" copies of these game-generated events (discussed in 3.2).

3.2 Data Processing

Given large-scale datasets of fine-grained game events have been collected and recorded, additional processing of the data may be performed at any time and for purposes of any analysis. By cleanly separating logging from processing and inference, we minimize the risk of changes to the game being required as research needs grow and change. However, there are difficulties to address with processing event data at scale; our primary concern is feature engineering. Feature engineering is often performed in an ad-hoc manner, tightly coupled to specifics of a given game's event logs, with little attention paid to repeatability of an analysis. This approach can lead to problems when attempting to replicate results over time or across similar games. Further, poorly-designed feature engineering processes can lead to prohibitive processing times on large-scale datasets. To address this, our architecture includes structures for a consistent, repeatable feature engineering system.

Generators This segment of the data processing architecture is responsible for the generation of post-hoc data, based on the game telemetry captured during data collection. The output of this processing infrastructure includes the following:

– Copies from the raw event data
– Newly-generated events, called "calculated" or "detector" events

– Session-level features
– Player-level features
– Population-level features.

Our data generation process uses small code modules, rather than monolithic implementations of ad-hoc, project-specific calculations. This directly enables support for reusable and reproducible analyses. Because of the choice to enforce a single event data schema in 3.1, these modules can be executed on any event data, even across games. While a small cost may be paid in initial implementation time, generator modules may be easily swapped in and out of any given data processing task, and may be applied to new data as it is collected.

Event Manager and Event Detectors The event manager is responsible for copying the raw input data for a distributable output dataset, and for passing the raw data to what we term "event detectors." An event detector is a small code module that accepts events as input and generates new events as output. For example, an "idle detector" might generate `begin_idle_behavior` and `end_idle_behavior` events based on long gaps in "meaningful" player interactions with the game, for some definition of "meaningful" chosen by a researcher. This definition could be based on an arbitrary threshold for gaps between any two game events, or on labels generated by a machine learning model, trained on human-coded observations of players.

In this way, our architecture supports post-hoc generation of game-referenced events with arbitrary complexity, from simple hand-written if-else logic to full machine learning models. This eliminates the need for logging clients to report anything other than objective, in-game actions and system responses.

Feature Manager and Feature Extractors The second component of data generation is feature extraction. The process here is similar to that described for event detectors. In this case, however, the individual modules are called "feature extractors," and each extractor generates a value summarizing the set of events it has observed, rather than new events. We do not place any specific constraints on the type of value generated by an extractor; the value could be boolean, numeric, a string, or even a JSON object, though numeric values tend to be most common.

The feature manager creates and tracks instances of each extractor module, and routes their outputs to the rest of the processing pipeline. This may include directing outputs to what we term "higher-order" feature extractors, which accept other features' values as inputs, in addition to event data. Extractors may operate at the level of a population, a player, or a gameplay session. Population-level features generate values based on all events in the given processing task, while session-level extractors use all events for a given session ID. In cases where a game provides user identifiers (discussed in ??), the architecture produces player-level extractors using all events for each given player ID. This creates further flexibility in the feature engineering process.

Processing Management Above the generators segment of our architecture sits a *processing manager*. This manager is responsible for organizing the common, high-level process of moving data through the pipeline, separating that process from the details of any given game's data or feature set. It maps inputs, in the form of raw event data, to the lower-level managers, described in the "Generators" section above. It also maps outputs into the calculated data storage. Finally, the processing manager is responsible for recording reports on the success or failure of processing tasks.

"Calculated Data" Storage Here, we distinguish storage of all data calculated post-hoc from raw event data logged from game clients. Unlike game telemetry data, in which events objectively occur once and are recorded for analysis, the inferences made about data are subject to change. This may be due to bugfixes during feature engineering, or changes in how an inference is made. By separating "calculated" data in our architecture, we allow such data to be governed by an invalidation scheme, which in turn allows re-calculation of data to be scheduled. The details of such a scheme are left to implementation, depending on researchers' needs for specific projects or games.

3.3 Data Access and Analysis

There are two primary concerns for access to data generated by the system. Namely, we must consider endpoints for the system, which deliver data in a specific format, as well as the analysis tools that will connect with those endpoints. While there are many technologies and formats that could theoretically be used, we identify several broad categories of tools for data exploration and analysis, not as a comprehensive summary but as a means to describe and account for common use cases. We then discuss four main endpoint types that cover a significant portion of use cases.

Due to the great variety of analysis tools and technologies for data access, this is the least prescriptive segment of our proposed architecture. Rather than present a specific set of relationships between components, we provide major categories of tools and show how different endpoints naturally map to different analysis tools, and how existing analysis pipelines can be improved by integrating our proposed architecture.

Analysis and Exploration Tools

– *Interactive Notebooks*
  Notebook environments organize executable code into individual "chunks" that can be independently edited and executed in any order. Jupyter Notebooks are a very popular implementation, supporting Python and other statistics-oriented programming languages in one environment [18]. Several other programming languages, including R and Matlab, provide their own notebook environments.
– *Data Dashboards*
  Dashboards are typically data visualization tools that display pre-set visual summaries of a selected dataset. Google's Looker Studio is a popular dashboard creation tool that integrates with Google-Cloud-based data stores.

Another commonly-used tool for dashboard design is Tableau, which provides general-purpose data visualization features.

– *System Monitors*
  This class of tool addresses the case of analyzing an analysis system itself. Many popular web service providers offer monitoring tools for their users, such as Amazon's CloudWatch or Microsoft's Azure Monitor. In the context of our proposed architecture, system monitors are generally assumed to be dedicated tools built as part of the architecture implementation.
– *Third-Party Tools*
  This last category serves as a catch-all for tools that do not fit cleanly into one of the identified categories. This may include custom-built scripts, offline data visualization and exploration tools, or other data mining and modeling tools. Examples here include Weka, a tool implementing many machine learning algorithms, and RapidMiner, a data mining and analytics tool.

Data Access Endpoints As noted, the variety of data tools and analysis needs make it impossible to definitively assign methods of data access to analysis tools. However, it is possible to identify cases where affordances and needs match well.

TSV Files A simple, flat, tabular file format is perhaps the most general and widely-supported means of data access possible. The comma-delimited version of such files (CSV) is supported by every major data processing software. Most support tab-delimited (TSV) files as well; we chose the TSV format because event and feature data structures contain some internal JSON-formatted elements. A tab delimiter ensures file parsers do not confuse the comma-separated JSON elements with the rest of the tabular data structure. The general-purpose nature and wide support for this file format format makes it a good match for general 3rd-party tools, as well as interactive notebook environments.

File API While TSV files may be easily shared and used locally, effective data sharing may be supported via an API for accessing datasets on a file server. Such an API could be used by a central repository website for dataset distribution. Accessing files from a server, rather than locally from the analyst's machine, may also be a good match for a notebook environment. Notebooks are often hosted in cloud-based environments, such as Google Colab or GitHub Codespaces. A simple API request that can obtain a specific dataset from any environment could maximize interoperability between cloud-based and locally-run notebooks. Certain types of dashboard are also well-served by a file API, specifically dashboards designed for summarizing a full dataset.

Random Access API A "random access API" allows users to request an arbitrary time range or set of IDs for data retrieval, rather than a fixed dataset as offered by a "File API." This is desirable for analyses that require data from a highly specific cohort or sub-population of players, such as a group from a specific gameplay event. This endpoint is a good match for dashboards designed for exploratory data analysis. Fixed datasets may be too limiting for such tools; instead, the ability to re-parameterize the choice of data on-the-fly is desirable.

Real-Time API This type of API provides access to data for users with active play sessions. The emphasis here is on low latency, allowing for evaluation of game data as it is generated. While this is the least general option we have listed, the implementation of such an API can unlock research that would not be feasible with any of the other data access methods described so far, such as studies of how teachers can use gameplay data to better facilitate classroom play sessions [29]. Real-time APIs are the best option for a system monitor application, allowing issues with the system to be detected as soon as they occur. They are also appropriate for certain data dashboard use-cases, particularly for dashboards meant for live usage concurrent with an event, such as a play-test.

4 Implementation

Having laid out a comprehensive description of a general architecture for repeatable, flexible, scalable processing of video game data, we now present an implementation of this architecture through a series of publicly-available, permissively-licensed software packages. We discuss the design decisions made to generate a specific implementation of the general architecture, and suggest alternate approaches or potential future improvements. Collectively, this discussion should serve to illustrate how the proposed architecture can be realized in a practical research environment.

4.1 Data Collection

The first block of the architecture is implemented in client packages `opengamedata-unity` and `opengamedata-js-log`, and logging package `opengamedata-logger`. The client packages implement the event schema proposed by Gagnon and Swanson [4] for Unity- and JavaScript-based games, and use simple configurations to direct their output to an instance of `opengamedata-logger`. Data is sent via standard HTTP POST requests. Request header parameters are used for any schema elements that are constant within a given gameplay session (e.g. identifier and versioning items), while data that varies per-event (game state, timestamps, etc.) is merged into a binary-encoded request body package. This allows for packaging of co-occurring events into a single request.

Conversely, `opengamedata-logger` decomposes the request body into multiple events, copying the request parameters into the corresponding elements of each event. We use a relational database system for raw event storage, and the packaging of events within requests is extended to allow multiple events to be packaged into a single database insertion. This minor implementation feature helps to manage potential server load issues by reducing the overall number of HTTP requests needed to log a given set of events.

Finally, we provide system monitoring via a web-sockets-based API, in the `opengamedata-monitor` package. This API serves a simple client web page for monitoring incoming data. Each event received by a `opengamedata-logger` instance is forwarded to an HTTP endpoint for the API, where the `opengamedata-monitor` instance parses and routes the events to its web clients based on per-game "rooms." Thus, incoming data for any game can be checked by researchers or system developers who need to ensure a particular game's logging is not failing.

4.2 Data Processing

A full implementation of the data processing step is contained in `opengamedata-core`. This implementation uses a request format that allows for users to specify either a date range or set of player/session IDs for processing, as well as a list of event detectors and feature extractors to be included in the output of the processing request. This allows users a great degree of control over the amount of data and complexity of processing. If a researcher is working with a particularly large-scale dataset, they can easily turn off extraction of any features they do not intend to use in their analysis, avoiding potentially long processing times.

Event Manager and Detectors In our implementation, an event detector defines an *event filter*, *update rule*, *trigger*, and *generator rule.* The event filter defines which game events are inspected by the detector. The manager passes any events that match the filter, one-by-one, to the update rule, which updates the detector's internal state. Following each update, the trigger checks for a condition in the detector state, and if the condition is met, "triggers" the generator rule, which produces a new event.

In the simplest case, a generator's update rule might contain hard-coded checks for patterns of event sequences. More complex detectors could use machine learning models trained to predict a human-coded label from sequences of event data. Thus, our implementation supports the broad set of use cases described by our proposed architecture.

Feature Manager and Extractors Like event detectors, our implementation of feature extractors use an *event filter*, *update rule*, and *generator rule*, which converts the current state into the single summary value. Unlike a detector, there is no need for a "trigger;" the feature manager simply invokes the generator rule when all events have been passed through the update rule.

In order to reduce the need for redundant calculations, our implementation also supports "sub-features." These are additional feature values generated by a single code module. Thus, a feature extractor module for an "average score" feature, which divides a total score by number of completed levels, could generate a "total score" sub-feature, avoiding redundant calculation by a standalone "total score" module.

Finally, our implementation supports a "second-order feature" class, as initial work towards the general "higher-order features" described in 3.2. These are feature extractors that define an additional *feature filter* and *feature update rule*, which function similarly to the event filter and update rule, but filter and update based on feature data.

As with the event detectors, our implementation supports arbitrarily complex calculation of feature values from event (and feature) data. Thus, our framework for feature engineering maintains flexibility and power, while ensuring analyses are reusable with future datasets that share the common event schema.

4.3 Data Access and Analysis

We have discussed the variety of data access and analysis approaches possible. Our implementation of the third block of the architecture provides packages for several data access services.

- The `opengamedata-core` package produces TSV output files by default. We automate the production of per-game, per-month datasets, including raw event, raw + calculated event, session feature, player feature, and population feature files. Thus, workflows for general 3rd-party tools and notebook environments are well-supported.
- The `opengamedata-api-files` package is a RESTful API for access to datasets stored on a file server. `OGDUtils` provides functionality for accessing data from the `opengamedata-api-files` API directly in any Python environment, such as a Jupyter Notebook.
- The `opengamedata-api-data` package implements a RESTful API for generalized requests for feature data. Users can request, per game, a custom set of features at the session, player, or population level, based on a set of IDs or a range of dates. The API server retrieves and extracts all feature data for the custom dataset using the `opengamedata-core` package, returning the data in a JSON-formatted web response.
- A general-purpose real-time API for data access remains a work-in-progress; the `opengamedata-monitor` package implements a sockets-based system monitoring tool as proof-of-concept for such an API. Future work on the API will allow a persistent instance of the monitor to build metrics and player models for select users as new events are generated live in a play-testing or classroom environment.

5 Case Studies

In order to illustrate the potential uses of Open Game Data we provide a few case studies in two categories, namely research using large datasets, and automated analysis designed for operation at large scales.

One of the promises of learning engineering is the ability to conduct large scale design experiments. This process was recently demonstrated in a four-condition experiment on the game Jo Wilder and the Capitol Case. This project utilized the existing game, making only low cost modifications to explore the effect of different game scripts on player engagement, enjoyment, and relatability with the game's protagonist with and nearly 12,000 game sessions [2,23].

In another recent experiment, researchers focused on understanding players strategies and approaches to a city-building game called *Lakeland*. In this study, data from 32,000 game sessions was analyzed using an unsupervised clustering method and visualized as radar plots to describe different types of player interactions [28].

Most recently, researchers utilized large datasets from 10 different games on Open Game Data to empirically validate their development of an interoperable ontology of game-based assessment metrics [8].

Another area that may benefit from our system is the automation of qualitative analysis. Utilizing a method previously developed for automated labeling of data from a cognitive tutor [21], researchers used game data from the game *Wake: Tales from the Aqualab* to train an automated detector for students who are experiencing "struggle" [14]. This was done by displaying segments of gameplay in text, effectively creating a textual narration of the players' actions and progression events. Researchers reviewed and labeled thousands of these game play text replays, providing a training set to a machine learning classifier. Once trained, the system can be deployed at arbitrarily large scale, and assess play sessions in real time.

6 Discussion and Future Work

Open Game Data is promising open science project in service of the educational game research community. The platform provides open source client-side code for integration into any game project, and a scalable architecture for data ingestion, real-time relay, and storage. A modular processing architecture supports the generation of post-hoc event identification, as well as feature engineering. Features can operate at different levels of aggregation, from a single play session up to a whole population. Additionally, these features can inform one another, so a session feature can be contextualized within a population, even with very large datasets. Finally, the data access components make data available via flat .tsv files as well as web APIs to integrate with extant and future tools.

While the project has shown promise to integrate with a number of games and support various forms of research, it has limitations across each component of the technical architecture and even more in the social infrastructure that will require significant additional investments and collaboration to resolve.

One of the fundamental challenges in games research is the significant cost of developing games that create the contexts, potential actions and data reporting required to conduct a specific experiment. One future direction for this project is to develop socio-legal-technological systems to broker the connections between researchers and the game studios that either own the game's licence or work within the game's licensing agreements. This will provide opportunities for existing games to be modified to support new research.

Similarly, additional infrastructure to support the automation of game modifications, or "remote configuration," is technically feasible and should be explored. If such infrastructure were developed, participating game studios could design their games to allow application of customizations at runtime without any modification to the game's compiled code. Establishing this capacity would allow for automation of experiments, where game owners have only to approve experiments before they are deployed.

Further work on data standards is also an area for development. Some preliminary work in this direction has been proposed by Gagnon et al. [5], but further work is needed to expand upon their proposed "ontologies" of event and feature collections. At each stage of coordination and standard development, new efficiencies are be gained for processes later in the data pipeline. The challenge will be developing these standards so they respect the flexibility of game design and do not add additional burden to implementation.

There are also many opportunities for reusable replay and annotation tools. Prior work (e.g. Rowe et al. [19]) have demonstrated the value of replay and labeling capability for learning research and game design. Work is underway to develop web-based tools to replay VR game sessions at high frame rates so that qualitative research can be conducted during or after a play session occurs. Given sufficient resources, a generalized client library could be produced that would create parity between events that are sent and events that would be required for replay and annotation to train detectors and other forms of models.

Another future technical direction focuses on the real time analysis of incoming game data. A first application of this capacity could be the creation of generalized tools for supporting classroom instruction with educational games, scaling the exploratory work of existing dashboard-based projects [29] and mixed reality interfaces for teachers [9]. The second application is for real-time augmentation of qualitative research efforts. This method, currently being explored by Hutt et al. [10] uses machine learning detectors to direct the attention of classroom researchers to previously-defined moments of interest so they can conduct observations and interviews at particular moments and synchronize those observations with game data. This is a promising new direction for human-machine pairing in educational research.

Acknowledgment. The authors would like to acknowledge Erik Harpstead for his recommendations and advice throughout the development of this project.

Compliance with Ethical Standards

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. El-Nasr, M., Drachen, A., Canossa, A.: Game Analytics: Maximizing the Value of Player Data. Springer (2013)
2. Gagnon, D.J., et al.: Exploring players' experience of humor and snark in a grade 3–6 history practices game. In: GLS 13.0 Conference Proceedings. Irving, CA (2022)
3. Gagnon, D.J., Ponto, K., Verbeke, M., Nathan, M., Kopp, K., Tredinnick, R.: Waddle: developing empathy for adélie penguins by direct embodiment in virtual reality. In: Joint International Conference on Serious Games, pp. 227–233. Springer (2023)

4. Gagnon, D.J., Swanson, L.: Open game data: a technical infrastructure for open science with educational games. In: Haahr, M., Rojas-Salazar, A., Göbel, S. (eds.) Serious Games, pp. 3–19. Springer Nature Switzerland, Cham (2023)
5. Gagnon, D.J., Swanson, L., Harpstead, E.: Defining an open data pipeline and standards for educational data mining and learning analytics with video game data. In: IEEE Conference on Games, in press (2024)
6. Gardner, J., Brooks, C., Andres, J.M., Baker, R.S.: MORF: a framework for predictive modeling and replication at scale with privacy-restricted MOOC data. In: 2018 IEEE International Conference on Big Data (Big Data), pp. 3235–3244 (2018). https://doi.org/10.1109/BigData.2018.8621874
7. Gee, J.P.: Learning and games. In: The Ecology of Games: Connecting Youth, Games, and Learning, pp. 21–40. The John D. and Catherine T. MacArthur Foundation Series on Digital Media and Learning (2007). http://www.mitpressjournals.org/doi/abs/10.1162/dmal.9780262693646.021
8. Gomez, M.J., Ruipérez-Valiente, J.A., Clemente, F.J.G.: Developing and validating interoperable ontology-driven game-based assessments. Expert Syst. Appl. **248**, 123370 (2024). https://doi.org/10.1016/j.eswa.2024.123370
9. Holstein, K., McLaren, B.M., Aleven, V.: Student learning benefits of a mixed-reality teacher awareness tool in AI-enhanced classrooms. In: Lecture Notes in Computer Science, vol. 10947, pp. 154–168. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-93843-1_12
10. Hutt, S., et al.: Quick red fox: an app supporting a new paradigm in qualitative research on AIED for STEM. In: Chapman & Hall/CRC Artificial Intelligence and Robotics Series, 1st edn. CRC Press, Boca Raton, FL (2023)
11. Ioannidis, J.P.A.: Why most published research findings are false. PLoS Med. **2**(8), e124 (2005). https://doi.org/10.1371/journal.pmed.0020124
12. Kim, Y.J., Almond, R.G., Shute, V.J.: Applying evidence-centered design for the development of game-based assessments in physics playground. Int. J. Test. **16**(2), 142–163 (2016). https://doi.org/10.1080/15305058.2015.1108322
13. Koedinger, K.R., Baker, R., Cunningham, K., Skogsholm, A., Leber, B., Stamper, J.: A Data Repository for the EDM Community: The PSLC DataShop, p. 21. CRC Press, Boca Raton, FL (2010)
14. Liu, X., et al.: Struggling to detect struggle in students playing a science exploration game. In: Companion Proceedings of the Annual Symposium on Computer-Human Interaction in Play, pp. 83–88. ACM, Stratford, ON, Canada (2023). https://doi.org/10.1145/3573382.3616080
15. Nielsen, M.: Reinventing discovery - the new era of networked. Science (2011). https://doi.org/10.1515/9781400839452
16. Nosek, B.A., et al.: Promoting an open research culture. Science **348**(6242), 1422–1425 (2015)
17. Nosek, B.A., Spies, J.R., Motyl, M.: Scientific utopia: II. Restructuring incentives and practices to promote truth over publishability. Perspect. Psychol. Sci. **7**(6), 615–631 (2012). https://doi.org/10.1177/1745691612459058
18. Pimentel, J.F., Murta, L., Braganholo, V., Freire, J.: A large-scale study about quality and reproducibility of Jupyter notebooks. In: 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), pp. 507–517. IEEE (2019)
19. Rowe, E., et al.: Advancing research in game-based learning assessment: tools and methods for measuring implicit learning, pp. 99–123. IGI Global (2020). https://doi.org/10.4018/978-1-7998-1173-2.ch006

20. Ruiperez-Valient, J.A., Kim, Y.J., Baker, R.S., Martinez, P.A., Lin, G.C.: The affordances of multivariate Elo-based learner modeling in game-based assessment. IEEE Trans. Learn. Technol. 1–14 (2022). https://doi.org/10.1109/TLT.2022.3203912
21. Baker, R.S., de Carvalho, A.: Labeling student behavior faster and more precisely with text replays. In: Educational Data Mining, Montréal, Québec, Canada (2008)
22. Scianna, J., Gagnon, D., Knowles, B.: Counting the game: visualizing changes in play by incorporating game events. In: Ruis, A., Lee, S. (eds.) Advances in Quantitative Ethnography. ICQE 2021. Communications in Computer and Information Science, vol. 1312. Springer, Cham, Malibou, CA (2021). https://doi.org/10.1007/978-3-030-67788-6_15
23. Slater, S., Baker, R.S., Gagnon, D.J.: Changing students' perceptions of a history exploration game using different scripts. In: Proceedings of the 30th International Conference on Computers in Education. Kuala Lumpur, Malaysia (2022)
24. Slater, S., Bowers, A., Kai, S., Shute, V.: A typology of players in the game physics playground. In: Proceedings of the 2017 DiGRA International Conference, p. 12 (2017)
25. Steinkuehler, C., Duncan, S.: Scientific habits of mind in virtual worlds, pp 1–14 (2008)
26. Steinkuehler, C.A.: Massively multiplayer online video gaming as participation in a discourse. Mind Cult. Act. **13**(1), 38–52 (2006). https://doi.org/10.1207/s15327884mca1301_4
27. Stenerson, M.E., Salmon, A., Berland, M., Squire, K.: ADAGE: an open API for data collection in educational games. In: Proceedings of the First ACM SIGCHI Annual Symposium on Computer-Human Interaction in Play, pp. 437–438. ACM, Toronto, Ontario, Canada (2014). https://doi.org/10.1145/2658537.2661325
28. Swanson, L., et al.: Leveraging cluster analysis to understand educational game player styles and support design. In: GLS 13.0 Conference Proceedings (2022)
29. Swanson, L., Gagnon, D.J., Scianna, J.: A pilot study on teacher-facing real-time classroom game dashboards. East Lansing, MI (2022)
30. Willinsky, J.: The Access Principle: The Case for Open Access to Research and Scholarship. Digital Libraries and Electronic Publishing, MIT Press, Cambridge, MA (2006)