ETS®

Measuring the Power of Learning.®

Research Report

ETS RR–16-10

Taming Log Files From Game/Simulation-Based Assessments: Data Models and Data Analysis Tools

Jiangang Hao

Lawrence Smith

Robert Mislevy

Alina von Davier

Malcolm Bauer

June 2016

ETS Research Report Series

Since its 1947 founding, ETS has conducted and disseminated scientific research to support its products and services, and to advance the measurement and education fields. In keeping with these goals, ETS is committed to making its research freely available to the professional community and to the general public. Published accounts of ETS research, including papers in the ETS Research Report series, undergo a formal peer-review process by ETS staff to ensure that they meet established scientific and professional standards. All such ETS-conducted peer reviews are in addition to any reviews that outside organizations may provide as part of their own publication processes. Peer review notwithstanding, the positions expressed in the ETS Research Report series and other published accounts of ETS research are those of the authors and not necessarily those of the Officers and Trustees of Educational Testing Service.

The Daniel Eignor Editorship is named in honor of Dr. Daniel R. Eignor, who from 2001 until 2011 served the Research and Development division as Editor for the ETS Research Report series. The Eignor Editorship has been created to recognize the pivotal leadership role that Dr. Eignor played in the research publication process at ETS.

ETS Research Report Series ISSN 2330-8516

RESEARCH REPORT

Taming Log Files From Game/Simulation-Based Assessments: Data Models and Data Analysis Tools

Jiangang Hao, Lawrence Smith, Robert Mislevy, Alina von Davier, & Malcolm Bauer

Educational Testing Service, Princeton, NJ

Extracting information efficiently from game/simulation-based assessment (G/SBA) logs requires two things: a well-structured log file and a set of analysis methods. In this report, we propose a generic data model specified as an extensible markup language (XML) schema for the log files of G/SBAs. We also propose a set of analysis methods for identifying useful information from the log files and implement the methods in a package in the Python programming language, glassPy. We demonstrate the data model and glassPy with logs from a game-based assessment, SimCityEDU.

Keywords data model; log file; process data; game- or simulation-based assessment

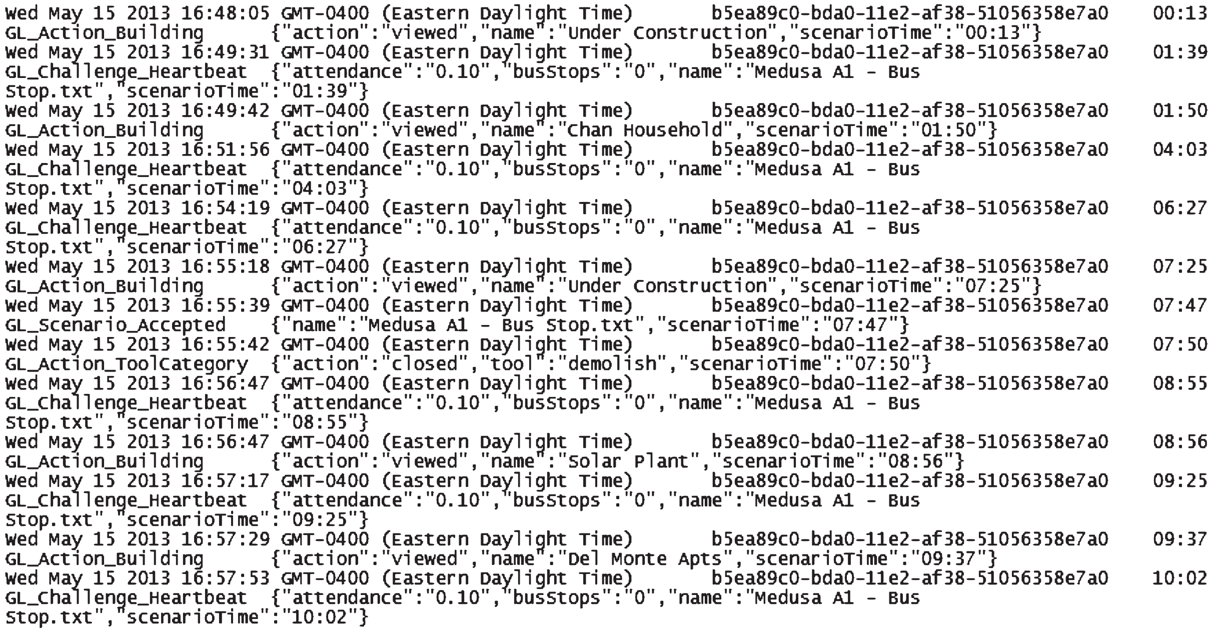
doi:10.1002/ets2.12096

Game/simulation-based assessment (G/SBA) has a number of advantages over traditional assessment for some purposes and is widely considered an important future direction for assessments (Mislevy et al., 2014). Evidence-centered design (ECD; Mislevy & Haertel, 2006) provides a framework for designing game-based assessment around a validity argument that connects a player's activities (evidence) to performance on some predefined construct(s) of interest. However, identifying the evidence from the log files becomes very inefficient if the information from the G/SBAs is not properly recorded into the log files. In practice, a log file is generally designed and developed by game/simulation developers whose primary interest is to debug the software system itself. For psychometrician working with G/SBAs, the goal is to find evidence that supports the measurements of certain constructs. These are distinct purposes; data and data structures that are optimal for one purpose may serve the other purpose poorly. Therefore, systematically rethinking how to structure the log files of G/SBAs is very important. It is worth mentioning that careful rethinking about the structuring of data files is important not only for G/SBAs but also for more traditional multiple-choice or constructed-response items (Bejar, Mattson, Wagner, Driscoll, & Hakkinen, 2014).

Here, we report our work on developing a data model for the log files of G/SBAs implemented in extensible markup language (XML; Hao, Smith, Mislevy, & von Davier, 2014) and a software package (library) in the Python programming language that provides basic functionalities for analyzing players' activities. The data model is enforced through an XML schema to ensure compliance with the data model and also to eliminate any potential erroneous entries in the log files. For the Python package, we built in some frequently used methods for analyzing the process data from log files, which facilitates the process of researchers developing their own analysis functionalities as extensions. Such a software package liberates psychometricians from the nontrivial logistics of handling the log files and querying for needed information and enables them to focus their work on designing scoring rubrics for the game/simulation events and the underlying measurement models, leading to more efficient design iterations (e.g., using the ECD framework described by Mislevy, Behrens, DiCerbo, & Levy, 2012 and von Davier & Mislevy, in press).

The report is organized as follows. In the second section, we review the current status of the log files from existing games/simulations made for assessments. In the third section, we introduce a data model for game log files and its implementation in XML. In the fourth section, we introduce analysis methods for game/simulation logs. In the fifth section, we introduce the specific design of the glassPy package and its implementation in the Python programming language. In the sixth section, we show some examples of implementing the data model and use the glassPy functionalities with data from the SimCityEDU[1] game (Mislevy et al., 2014). In the last section, we summarize the findings of our report and outline limitations and future development. We include the current XML schema for the data model in the Appendix.

Corresponding author: J. Hao, E-mail: jhao@ets.org



```
Wed May 15 2013 16:48:05 GMT-0400 (Eastern Daylight Time)      b5ea89c0-bda0-11e2-af38-51056358e7a0    00:13
GL_Action_Building      {"action":"viewed","name":"Under Construction","scenarioTime":"00:13"}
Wed May 15 2013 16:49:31 GMT-0400 (Eastern Daylight Time)      b5ea89c0-bda0-11e2-af38-51056358e7a0    01:39
GL_Challenge_Heartbeat  {"attendance":"0.10","busStops":"0","name":"Medusa A1 - Bus
Stop.txt","scenarioTime":"01:39"}
Wed May 15 2013 16:49:42 GMT-0400 (Eastern Daylight Time)      b5ea89c0-bda0-11e2-af38-51056358e7a0    01:50
GL_Action_Building      {"action":"viewed","name":"Chan Household","scenarioTime":"01:50"}
Wed May 15 2013 16:51:56 GMT-0400 (Eastern Daylight Time)      b5ea89c0-bda0-11e2-af38-51056358e7a0    04:03
GL_Challenge_Heartbeat  {"attendance":"0.10","busStops":"0","name":"Medusa A1 - Bus
Stop.txt","scenarioTime":"04:03"}
Wed May 15 2013 16:54:19 GMT-0400 (Eastern Daylight Time)      b5ea89c0-bda0-11e2-af38-51056358e7a0    06:27
GL_Challenge_Heartbeat  {"attendance":"0.10","busStops":"0","name":"Medusa A1 - Bus
Stop.txt","scenarioTime":"06:27"}
Wed May 15 2013 16:55:18 GMT-0400 (Eastern Daylight Time)      b5ea89c0-bda0-11e2-af38-51056358e7a0    07:25
GL_Action_Building      {"action":"viewed","name":"Under Construction","scenarioTime":"07:25"}
Wed May 15 2013 16:55:39 GMT-0400 (Eastern Daylight Time)      b5ea89c0-bda0-11e2-af38-51056358e7a0    07:47
GL_Scenario_Accepted    {"name":"Medusa A1 - Bus Stop.txt","scenarioTime":"07:47"}
Wed May 15 2013 16:55:42 GMT-0400 (Eastern Daylight Time)      b5ea89c0-bda0-11e2-af38-51056358e7a0    07:50
GL_Action_ToolCategory  {"action":"closed","tool":"demolish","scenarioTime":"07:50"}
Wed May 15 2013 16:56:47 GMT-0400 (Eastern Daylight Time)      b5ea89c0-bda0-11e2-af38-51056358e7a0    08:55
GL_Challenge_Heartbeat  {"attendance":"0.10","busStops":"0","name":"Medusa A1 - Bus
Stop.txt","scenarioTime":"08:55"}
Wed May 15 2013 16:56:47 GMT-0400 (Eastern Daylight Time)      b5ea89c0-bda0-11e2-af38-51056358e7a0    08:56
GL_Action_Building      {"action":"viewed","name":"Solar Plant","scenarioTime":"08:56"}
Wed May 15 2013 16:57:17 GMT-0400 (Eastern Daylight Time)      b5ea89c0-bda0-11e2-af38-51056358e7a0    09:25
GL_Challenge_Heartbeat  {"attendance":"0.10","busStops":"0","name":"Medusa A1 - Bus
Stop.txt","scenarioTime":"09:25"}
Wed May 15 2013 16:57:29 GMT-0400 (Eastern Daylight Time)      b5ea89c0-bda0-11e2-af38-51056358e7a0    09:37
GL_Action_Building      {"action":"viewed","name":"Del Monte Apts","scenarioTime":"09:37"}
Wed May 15 2013 16:57:53 GMT-0400 (Eastern Daylight Time)      b5ea89c0-bda0-11e2-af38-51056358e7a0    10:02
GL_Challenge_Heartbeat  {"attendance":"0.10","busStops":"0","name":"Medusa A1 - Bus
Stop.txt","scenarioTime":"10:02"}
```

Figure 1 Log file from the pilot study of SimCityEDU (Mislevy et al., 2014).

Current Status of Log Files From G/SBAs

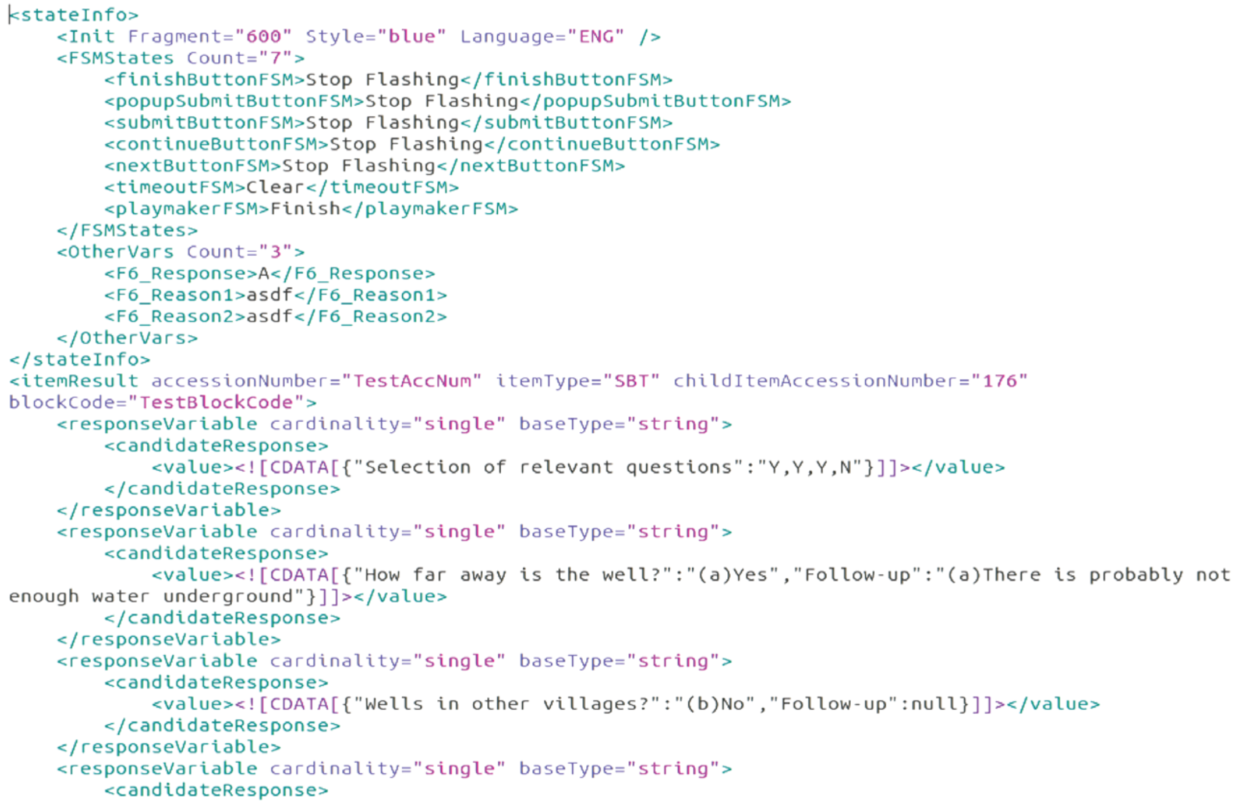
Activities are captured in two ways in existing G/SBAs: (a) activities are recorded into log files or (b) game/simulation information is ported directly into a database in real time. Although the focus in this report is mainly on the log file approach, the report also provides valuable guidance for designing an appropriate database schema to store information. The log files of existing G/SBAs can be classified into two categories: (a) plain text files with line-by-line dumps of events without systematic structures and (b) structured logs in XML, JavaScript Object Notation (JSON), or some other structured format. Figures 1 and 2 show two example log files corresponding to these categories.

In the case of Figure 1, because the log file is not structured, parsing through the text file and extracting information is cumbersome and error-prone, though not impossible. For example, not all lines have the same number of elements, and the number of logical chunks in each line is not fixed. Writing a parser to extract information from such a log requires daunting effort and presents an unpredictable number of exceptions. In the case of Figure 2, although the log file is in a structured XML format, its organization is not optimized for analysis. For example, the analysis unit for the log is not clearly represented, and the attributes and elements are not systematically organized. These characteristics make working with the potential evidence contained in the log files inefficient and error-prone.

In current practice, researchers in the field of educational data mining normally create their own conventions for extracting information from log files based on their own convenience (Kerr & Chung, 2012). If the log files come with erroneous entries and are not well formatted, extracting information in this way is very inefficient, as a user then needs to include many exceptions in his or her programs to accommodate the potential erroneous entries and irregular structures. Moreover, the user additionally needs to customize his or her reduction pipeline for different games/simulations.

Educational game/simulation developers, conversely, generally develop their own data collection systems, on top of which some analysis tools are built to provide some predefined and high-level aggregated information and visualization (Gibson & Jakl, 2015; Halverson & Owen, 2014). Although intended for researchers, these tools are not sufficient, as researchers normally need to manipulate in-game activities at a much finer level (e.g., for each time stamp) and craft their own ways of slicing, aggregating, and visualizing the data. All these processing stages need to be done at a faster turnaround rate rather than waiting for game developers to implement (if they eventually agree) the needed functionalities into their system.

Can we develop a data model for log files that is well structured and can be applied to a wide range of G/SBAs? Although specific G/SBAs can be very different, they share many commonalities that provide the grounds for developing a data



```
<stateInfo>
    <Init Fragment="600" Style="blue" Language="ENG" />
    <FSMStates Count="7">
        <finishButtonFSM>Stop Flashing</finishButtonFSM>
        <popupSubmitButtonFSM>Stop Flashing</popupSubmitButtonFSM>
        <submitButtonFSM>Stop Flashing</submitButtonFSM>
        <continueButtonFSM>Stop Flashing</continueButtonFSM>
        <nextButtonFSM>Stop Flashing</nextButtonFSM>
        <timeoutFSM>Clear</timeoutFSM>
        <playmakerFSM>Finish</playmakerFSM>
    </FSMStates>
    <OtherVars Count="3">
        <F6_Response>A</F6_Response>
        <F6_Reason1>asdf</F6_Reason1>
        <F6_Reason2>asdf</F6_Reason2>
    </OtherVars>
</stateInfo>
<itemResult accessionNumber="TestAccNum" itemType="SBT" childItemAccessionNumber="176"
blockCode="TestBlockCode">
    <responseVariable cardinality="single" baseType="string">
        <candidateResponse>
            <value><![CDATA[{"Selection of relevant questions":"Y,Y,Y,N"}]]></value>
        </candidateResponse>
    </responseVariable>
    <responseVariable cardinality="single" baseType="string">
        <candidateResponse>
            <value><![CDATA[{"How far away is the well?":"(a)Yes","Follow-up":"(a)There is probably not
enough water underground"}]]></value>
        </candidateResponse>
    </responseVariable>
    <responseVariable cardinality="single" baseType="string">
        <candidateResponse>
            <value><![CDATA[{"Wells in other villages?":"(b)No","Follow-up":null}]]></value>
        </candidateResponse>
    </responseVariable>
    <responseVariable cardinality="single" baseType="string">
        <candidateResponse>
```

Figure 2 Log file from the pilot study of the National Assessment of Educational Progress Technology and Engineering Literacy pump repair task (National Center for Education Statistics, 2013).

model that can be applied to generic log files. Fixing the data model will allow us to further develop a software package to handle some generic log file analyses. In the next section, we first propose such a data model.

Data Model for G/SBA Logs

Data Model

A *data model* describes how data are structured and defines the data types for different fields. Many different data models can be used to represent the same information in a G/SBA log file, depending on how a user determines the logical chunks and how much detail the user wants to include. Generally speaking, the elements in the log file of a G/SBA can be classified as *atomic* or *composite*. Atomic elements are the information that cannot be derived from other information in the log file, whereas composite elements are those elements that can be derived from other information in the logs. To specify a data model in the most concise yet complete way, we need to identify the atomic elements in a given game/simulation. All other composite elements can be derived later on in the processing stage. However, some composite elements may not be easily reconstructed based on atomic elements later on but are rather easy to record from the server side of the game/simulation. An example is the attempt number for a player who plays the game several times in a year. Although we can reconstruct the attempt number based on the number of times the player has played, handling from the client side the bookkeeping over all attempts over such a long time period will be very cumbersome and error-prone. It is much easier to assign an attempt number from the server side each time the player plays. In practice, we will treat these kinds of elements as atomic.

Our goal is to design a data model for the logs of games/simulations for educational assessment, not for all games/simulations. This constraint narrows the possibilities of the log files and allows us to create a generic data model. Focusing only on logs for G/SBAs allows two major simplifications. First, assessment is normally for an individual person so that we do not need to worry about interactions among multiple players. Second, ECD will help clarify the evidence sought from the game/simulation to support inferences about particular constructs. This will filter out many

Table 1 Attributes of Events

Attribute	Description
Event name	Element describing the name of the event
Event start time	Element describing the starting time of the event
Event end time	Element describing the ending time of the event
Event by	Element describing who triggers/commits the event
Event to	Element describing on whom the event has an effect
Event location	Element describing where the event happens
Event result	Element describing the results of the event

in-game/simulation activities that may not be useful for assessment purposes. Based on these considerations, the activities of the player and the game system in a G/SBA are essentially a collection of time-stamped events of different types, with different attributes and consequences. This makes it possible to develop a generic data model that can be applied to a number of G/SBAs.

To arrive at such a data model, we need to clarify the basic unit of analysis. As the assessment is for each individual player, at first blush, it might appear that the player would be the natural unit of analysis. But because each player can play the game/simulation many times, a more appropriate choice of unit of analysis would be each time each player plays the game/simulation, which we call a *session*. A single player ID, session ID, and attempt ID are the minimum requirements to specify a session. Each session consists of many events of different types. To specify an event at the minimal level, we need at least the attributes shown in Table 1.

So far, we have introduced the minimal set of attributes needed to describe a session and an event. In reality, there are different types of G/SBAs, and each type may have its additional unique features that need to be captured. Therefore, in addition to the minimal set of attributes, we should also allow users to add their additional attributes. First, at the session level, users may want to include more information about a given session, so we need to provide a placeholder for this extra information. Second, at the event level, users may have additional attributes to describe a given event, so we also need to provide a placeholder for this. As a result, aside from the attributes we set forth, we need to have two additional attributes, namely extended session data and extended event data, to accommodate the user-defined information. As we cannot predict the names and specifics for these user-specified fields in advance, we will use pairs of keys and values as placeholders; that is, each user-specified field is realized by a set of keys and the corresponding values. For example, if a user also wants to record the age of the player, the user can use

```
 key = "age," value = 20
```

Users can specify as many of these pairs as they want, and these pairs can be specified at both the session level and the event level. Such key and value pairs can be used to specify almost any complex attribute. Figure 3 is a schematic of the structure of the data model discussed.

So far, we have a general idea about the data structure that a log file from a G/SBA should follow. Next, we specify the data model in XML format and provide the corresponding XML schema for compliance checking (we introduce details of both in the next section). It is worth noting that we choose to use XML here, but the same data model can be realized using other structured languages, such as JSON, in a similar way.

Extensible Markup Language and Extensible Markup Language Schema

XML is a markup language that defines a set of rules for encoding documents in a format that is both human readable and machine readable (Bray, Paoli, Sperberg-McQueen, Maler, & Yergeau, 1998). XML is self-explanatory and is fully specified by the markup constructs. It has been widely used for data standardization in science (e.g., Williams et al., 2002) and industry. A detailed description of XML is beyond the scope of the current report, but readers can refer to the W3C Web page for more details.[2]

An XML schema is a file that provides a detailed description of the corresponding XML file (with a well-defined structure of its own). It defines the structure of the XML file and specifies the data type of each field in the XML file. The XML schema is also written in XML and is a convenient way to specify the data model that the XML file follows. Most important, the schema can be used to validate whether an XML file complies with its definition. For example, if a certain attribute is defined to contain an integer, a value of “TRUE” will be flagged. This is an important feature for ensuring that a delivered XML file meets the requirements set forth by the data model.

Data Model for a Game Log in Extensible Markup Language

We now can specify the data model in Figure 3 in XML format. The XML skeleton corresponding to the data model is as follows:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<gameLog xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <session>
        <sessionID> ** </sessionID>
        <playerID> ** </playerID>
        <attemptID> ** </attemptID>
        <sessionExtData>
              <pair>
                  <key> ** </key>
                 <value> ** </value>
              </pair>
              ....
        <eventSequence>
            <event>
                <eventName> **</eventName>
                <eventStartTime> ** </eventStartTime>
                <eventEndTime> ** </eventEndTime>
                <eventBy> ** </eventBy>
                <eventTo> ** </eventTo>
                <eventResult> ** </eventResult>
                <eventLocation> ** </eventLocation>
                 <eventExtData>
                     <pair>
                          <key> ** </key>
                          <value> ** </value>
                     </pair>
                         ……
                 </eventExtData>
            </event>
            <event>
                    …
            </event>
            …
        </eventSequence>
    </session>
</gameLog>
```

Double asterisks (**) in the XML skeleton represent any string. Suspension points (…) indicate that the elements can be repeated in parallel. Table 2 details each tag in the XML skeleton.

Aggregation of Distributed Log Files

A log file will be generated when a player plays one session of the game/simulation. In a real administration of a G/SBA, a game/simulation can either be distributed on each player’s computer or hosted on a central server. Either means of administration will lead to a set of log files in XML format with root element < gameLog/>. During data reduction and analysis, it is customary to aggregate the log files into certain larger chunks to facilitate analysis—all the sessions of a given player, for example, or all first sessions of a group of players. The data model we specified makes it very easy to aggregate all log files simply by appending the contents enclosed by the < session/> element.

Analysis Methods

So far, we have introduced the data model for log files of G/SBAs. The next question is what analyses should we perform after generating a log file following the specified data model. Different games/simulations have their specific features, and

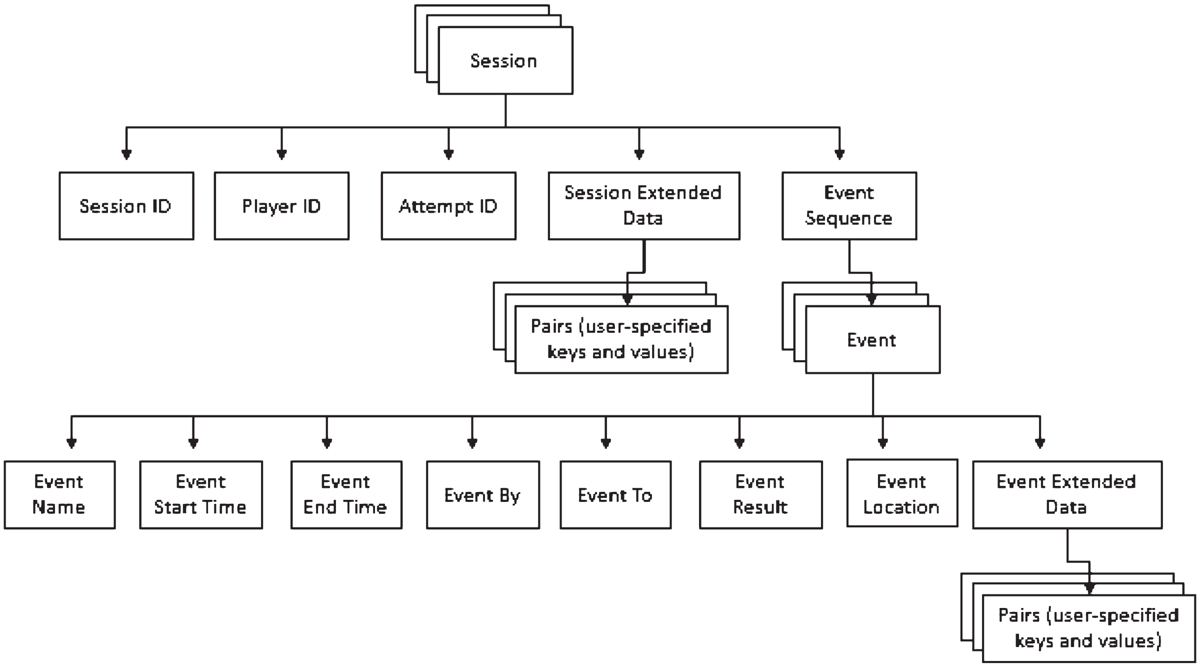


Figure 3 Schematic of the structure of our proposed data model for game/simulation logs.

Table 2 Description of the Tags in the Extensible Markup Language Skeleton

Tag Name	Meaning	Data Type[a]
<gameLog>	Root element of the game log file	idType
<session>	Child element of <gameLog>, specifying the block of session	
<sessionID>	Child element of <session>, specifying the session's name or other identifier	idType
<playerID>	Child element of <session>, specifying the player	idType
<attemptID>	Child element of <session>, specifying the attempt of the player for this session	idType
<sessionExtData>	Child element of <session>, specifying additional information defined by the users about the session	dictType
<eventSequence>	Child element of <session>, specifying the block of events in the given session	
<event>	Child element of <events>, specifying information about a specific event in the game/simulation	eventType
<eventName>	Child element of <event>, specifying the name of the specific event	idType
<eventStartTime>	Child element of <event>, specifying the time when the event starts	timestampType
<eventEndTime>	Child element of <event>, specifying the time when the event ends	timestampType
<eventBy>	Child element of <event>, specifying the committer of the event	idType
<eventTo>	Child element of <event>, specifying the target of the event	idType
<eventResult>	Child element of <event>, specifying the outcome of the event	String
<eventLocation>	Child element of <event>, specifying the location of the event; location can be physical or the progress location of the game/simulation	idType
<eventExtData>	Child element of <event>, specifying additional information about the event specified by users	dictType

[a] A detailed definition of each of the data types can be found in the schema in the Appendix.

analysts need to apply or develop specific analysis methods as needed. However, as we have shown, all games/simulations share some common characteristics, a fact that allows us to put forth some generic analysis methods in addition to the common data model. While developing the package, we envisioned two types of analyses for users of different levels of programming proficiency. First, for entry-level users, we provide an overall summary function that will generate summary information based on the log files. In this case, a user does not need to be a proficient Python programmer and can obtain much summary information about the log files simply by typing a single command (i.e., running a single function).

Second, for more advanced users, we developed a set of functions that can be called individually to perform specific analyses. In this case, users need to know how to program in Python and can incorporate the functions into their own analysis pipelines. In the following, we introduce the details of these functionalities.

LogSafari

This overall summary function is intended for entry-level users. Users can simply run this function to obtain a set of useful summary information about the log files. Following is a list of summary information we have currently implemented in the logSafari function: (a) check compliance against the data model schema, (b) report total number of sessions, (c) report number of unique players, (d) summarize number of attempts of each player, (e) report the starting time and duration of each session, (f) report the frequency of each event, (g) report the frequency of adjacent event pairs, and (h) create an event sequence number versus event time plot.

It is worth noting that this list is not static but expanding as our research is ongoing. We will include new summary information in this function as long as we think it is informative and generalizable.

Extensible Markup Language Validation

This functionality implements a compliance check of a delivered log file against the data model specified by our XML schema; that is, every log file that claims to follow the data model will be checked as to whether it meets the requirements specified by the schema. If it does not meet the schema's requirements, we need to notify the user to make changes to the log files to comply with the schema. This process is a bit painful for developers at first, but the benefit is worry-free data analysis later on. As long as the game developer can correctly specify the log file based on the schema we provided, this process will be smooth. Moreover, such a validation mechanism can safeguard the logs from potential erroneous entries that will plague the analysis later on due to unknown bugs in the logging system.

Event N-Gram

The core of a game/simulation is a sequence of events occurring during a session. Therefore, obtaining a general summary of information about the events and event sequence is generally the first step in analyzing the logs. We are interested not only in the frequency of each event (unigram) but also in the frequency of the event sequence (bigram, trigram, … , N-gram). If we temporarily put aside the temporal dependence and look only at the types of events in the logs, a complete description of the events can be casted as an N-gram expansion of the event sequences, that is, from unigram (frequency of different events) to bigram (frequency of two consecutive event sequences) up to N-gram (frequency of n consecutive event sequences). By appending the N-gram features, one can form vectors with elements as the frequency of the corresponding N-gram. With such a vector, one can classify and compare players' performances using different similarity measures. In the glassPy package, we provide a function that can calculate the frequency of N-gram events up to a user-specified n.

Event Sequence Matching

In a G/SBA, both the system state information and a player's activities are essentially a list of time-stamped events. The order and time stamps of the sequences or subsequences convey information about a player's proficiency on certain tasks. Therefore, these sequences and subsequences are important ingredients for scoring rules for the G/SBA. As such, it is customary to identify certain event subsequences from the whole event sequence. We developed a function to locate all specified subsequences from the whole sequence and return their indices in the whole sequence. This function greatly facilitates implementations of scoring rules based on the subsequences. It also simplifies further analysis of the time intervals between the events in a subsequence.

Weighted Levenshtein Edit Distance

Edit distance can be used as a measure to compare how different two event sequences are and has been shown to be useful for scoring G/SBAs (Hao, Shu, & von Davier, 2015) and for clustering performance (Bergner, Shu, & von Davier, 2014).

To choose an appropriate edit distance, one can change the editing operations or vary the weights assigned to each edit operation. The most widely used edit distance is the *Levenshtein distance* (Levenshtein, 1966), which defines the edit operations as deletion, insertion, and substitution, all with equal weight. If we fix the edit operation and let the weights vary, we arrive at the weighted Levenshtein distance (Jurafsky & Martin, 2000). For different G/SBAs, one can adjust the weights to achieve optimal separation of event sequences based on content knowledge about the specific game/simulation. In the glassPy package, we provide a function to calculate weighted Levenshtein edit distances using dynamic programing.

Hierarchical Vectorization

We have discussed the N-gram representation of game events, which ignores temporal information about the events. To capture temporal information, we introduce a hierarchical vectorization method. In this method, we put aside the specific events' names and focus only on the time stamps of all the events. That way, the temporal information is summarized by a set of time intervals of different lengths. The real challenge is how to align different players' sets of time intervals for comparison, as players have different play times and varied time intervals.

The method we propose here is to align different players' time intervals by ranking orders. We rank each player's time intervals from highest to lowest, that is, the longest, second longest, and so on. Then, we can compare across players by aligning the ranking orders; for example, we can compare the longest interval, second longest interval, and so on, across all players. This is equivalent to building a hierarchical clustering tree based on the time intervals using a single linkage, which is why we call it *hierarchical vectorization*. Each ranking order corresponds to a depth level of the hierarchical tree. At each depth level, we calculate some characteristic feature variables to represent the temporal information at each level. Two natural feature variables are the interval length and the location of corresponding intervals at the given depth level. In addition to these two feature variables, we also want to know how fragmented the intervals are at each depth level. To quantify this, we introduce two additional feature variables, namely, the *mean intracluster variance* and the *intercluster variance*. The former is defined as the average of the time interval variance within each cluster formed at the corresponding hierarchical level. The latter is defined as the variance of the time stamps corresponding to the cluster mean location. By appending the feature variables from different levels up to a specified depth level, a vector can be created to represent the temporal information for each single session. We have applied this method to analyzing play strategy in the SimCityEDU game (Hao & Kitchen, 2015) as well as the keystroke information of essay composition (Zhang, Hao, Li, & Deane, in press). In the glassPy package, we provide a function to calculate the hierarchical vectorization.

Time Interval Distribution of Consecutive Events

Consecutive events in game logs play important roles. Although the Markov assumption (i.e., an event depends only on the immediately preceding event) may not be true in general, the first-order association of the events clearly encodes most of the information. For consecutive events that appear many times, the time interval distribution will provide valuable information about the particular pairs of events. For example, when the time intervals between two consecutive events are very small for all the pairs, this suggests that little thinking is needed when executing the second event after the first. Conversely, if the variance of the interval distribution is large, this might imply that the player needs to think about what to do after the first event in different situations. As an illustrative example, think of our daily use of the mouse and keyboard. The time intervals between a left click and a mouse move should show little variance. However, the intervals between a keystroke on the keyboard and movement of the mouse may vary a lot. This is because in the latter case, there is more thinking about when to use the keyboard and when to use the mouse. Therefore, the distribution of the intervals between consecutive events will signify the importance of the event pairs and direct our attention to those interesting pairs. In the glassPy package, we provide a function to return the time interval distribution for a set of user-specified event pairs.

Recap

We listed here seven analysis methods. This list is far from complete and is currently expanding as our research proceeds. In the next section, we introduce the implementation of the methods in the Python programming language.

Table 3 Dependent Python Packages for glassPy

Library Name	Main Functionality	Package URL
Numpy	Backbone of all low-level numerical operations	http://www.numpy.org/
Scipy	Package that include many algorithms for scientific computing	http://www.scipy.org/
Pandas	Data organization, query, etc.	http://pandas.pydata.org/
Matplotlib	2-D plotting	http://www.matplotlib.org/
NLTK	Natural language processing	http://www.nltk.org/
Sklearn	Machine learning tool kit with most of the popular machine learning algorithms	http://www.scikit-learn.org/

Implementation in Python

The Python programming language is a widely used open source and interpretative language that is well suited for both interactive analysis and software development. It is one of the major programming languages used by major IT companies, such as Google and Facebook. Python contains many well-developed software packages for statistical analysis, data mining, text processing, visualization, and so on. Given the virtues of Python, we chose to use it to implement our analysis methods. In this section, we introduce some general ideas behind this implementation.

Pandas Data Frame

So far, we have defined a structured log file in XML format. However, when we analyze the logs, we need to read the XML logs into computer RAM and store them as a certain data structure so that we can operate on them directly. Python contains many different data structures. The one we chose to implement in our package is the data frame provided in the pandas package (McKinney, 2012), which is analogous to the data frame in the R programming language. The data frame is essentially a table structure with additional index columns (one or many). This data frame supports various analysis methods, such as query, slice, and join, making it an ideal choice for data structures in Python programming.

In the glassPy package, we provide a function "xml_to_dataframe" to convert an XML log file following our data model into a pandas data frame. When converting, it is necessary to clarify the data type in the data frame for each field or element of the data model. For most of the fields, this is straightforward. ID type corresponds to string type in Python, and integer type corresponds to integer type in Python. The only place that may be complicated is the dictionary type we used as a placeholder for user-specified information in the data model. The good news is that there is a built-in data type in Python, dictionary, that can be seamlessly mapped to the key–value pairs specified in the dict type. The columns of the data frame are a direct mapping of the tag names in the data model, except we change uppercase to lowercase and change camel case with underscore hyphenated to comply with the PEP-8 style guide widely used in Python programming (van Rossum, Warsaw, & Coghlan, 2001). For example, "eventName" (in camel case) will be changed to "event_name" (in PEP-8 style). Accordingly, the column names in the data frame are as follows: session_id, player_id, attempt_id, session_ext_data, event_name, event_start_time, event_end_time, event_by, event_to, event_location, event_ext_data.

Each row of the data frame corresponds to a time stamp of the event. This format means the session_id, player_id, attempt_id, and session_ext_data will be repeated throughout the same session for all events. This is clearly not an economical way to do things in the sense of memory usage. However, it is a compromise, because we cast everything into a table structure. If we were to choose a hierarchical data structure, we could avoid this. However, because many people in educational measurement use Microsoft Excel-based software, we kept a table structure in the current implementation.

Dependent Packages

The glassPy package is dependent on many other open source packages in Python. These dependencies are summarized in Table 3.

Operation of the Functions

After reading the log file into a pandas data frame, we need to apply various analysis methods to the data or subset of the data. One of the crucial parts of this process is choosing the subset of the data frame under different conditions. The most common subset is each column. Each column of the data frame can be accessed by the "."

operation. For example, df.player_id refers to the "player_id" column of a data frame called "df." To choose a subset that meets certain criteria, the pandas data frame provides a nice query function that can execute structured query language-type queries to choose the subset of the data. For example, if we want to choose the subset that includes all the events with the name "open the door" and belongs to player "bob," we can do the following query:

```
df.query('event_name == open the door & player_id == bob')
```

Once we have chosen the subset of the data, we are ready to apply the functions in the glassPy package. Specific ways to call the functions can be found in the documentation part of each function. In addition to the functions we supplied in the package, users can use all other functions provided by the vast collection of existing Python packages. For the specific usage of those functions, the reader can refer to the relevant documentation.

Summary of the Implementation

The glassPy package provides a suite of convenient functions to facilitate the analysis of log files from G/SBAs. Because it is written in Python, a user can easily incorporate the vast collection of Python packages for statistical analysis, machine learning, visualization, and so on. The set of functionalities in our package is expanding as our research proceeds.

Examples From SimCityEDU

In this section, we showcase examples using some of the analysis methods presented in previous sections to give readers a general sense about functionality. We use the log files from SimCityEDU, an educational game developed by GlassLab (Mislevy et al., 2014). This game is modified from a well-known video game, SimCity. The game has four different missions, School Is In, We Need Jobs, Pollution Problems, and It's Complicated, by which players' skills in system thinking, critical thinking, cause and effect, and system modeling are measured. For details about SimCityEDU, see Mislevy et al. (2014).

Log File Validation

The raw log file from SimCityEDU does not follow our data model, as the file was developed before the data model existed. For pedagogical purposes, we convert the raw log files from SimCityEDU into a format in XML that complies with the schema and name it *schema_test_simcityEDU.xml*. We also create another XML file, but with date/time recorded differently from the specification of our data model, and name it *schema_test_simcityEDU_incorrect.xml*. We show snippets of the two in Figures 4 and 5, respectively. Lines 20 and 21 are where different date/time formats are used.

Next, we show how to use the function xml_validation in the glassPy package to validate the XML file against the schema (see the Appendix). In Figure 6, we show a screenshot of the procedures.

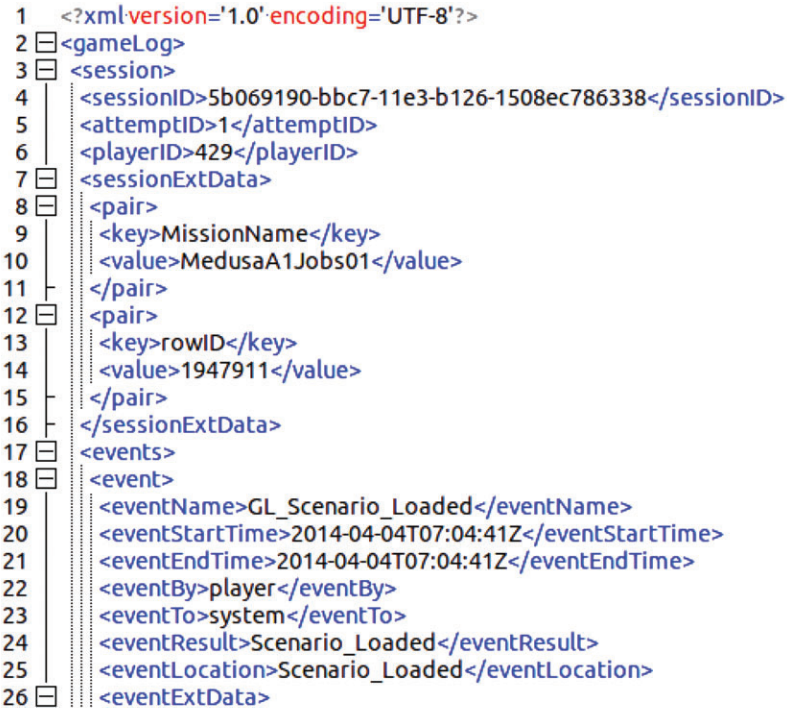
Event Frequency (Unigram) and Interevent Interval Distribution

As we pointed out earlier, we included some convenient functions in the glassPy package. In the following figures, we show two plots from two different analyses. The first one, Figure 7, concerns the frequency distribution of different events in one session of SimCityEDU.

Based on such a plot, one can quickly classify the events into different categories. For example, the first event, GL_Challenge_Heartbeat, has a much higher frequency than the others. This event is the system pull of the game state information, and GL_Zone to GL_Workers are events that are more frequent than the rest, as they are also system information pulls of a different kind. The rest of the events that happen at much lower frequencies are actions performed by the player. Although more careful scrutiny is needed to more meaningfully interpret these events, the plot in Figure 7 can help us quickly narrow our scope for future analysis.

In the second example (Figure 8), we show the distribution of interevent time intervals for the most frequent event pairs (bigrams) for a given session. This plot can help us identify the interesting event sequences at the bigram level, as discussed in the Analysis Methods section.

From Figure 8, one can observe that the variance of the time intervals between the event Simoleons and the event Zone is quite large compared to that between Coal_Power_Produced and Zone. The first pair are the variance of the thinking time between checking the available money (Simoleons) and planning for building power plants (Zone). The second pair are the variance of thinking time between checking the amount of energy produced (Coal_Power_Produced)



```
<?xml version='1.0' encoding='UTF-8'?>
<gameLog>
 <session>
  <sessionID>5b069190-bbc7-11e3-b126-1508ec786338</sessionID>
  <attemptID>1</attemptID>
  <playerID>429</playerID>
  <sessionExtData>
   <pair>
    <key>MissionName</key>
    <value>MedusaA1Jobs01</value>
   </pair>
   <pair>
    <key>rowID</key>
    <value>1947911</value>
   </pair>
  </sessionExtData>
  <events>
   <event>
    <eventName>GL_Scenario_Loaded</eventName>
    <eventStartTime>2014-04-04T07:04:41Z</eventStartTime>
    <eventEndTime>2014-04-04T07:04:41Z</eventEndTime>
    <eventBy>player</eventBy>
    <eventTo>system</eventTo>
    <eventResult>Scenario_Loaded</eventResult>
    <eventLocation>Scenario_Loaded</eventLocation>
    <eventExtData>
```

Figure 4 Snippet of schema_test_simcityEDU.xml, which fully complies with our data model.

and planning for building power plants (Zone). A valuable implication from this is that the information of actual power produced has more direct impact on this player's decision to build power plants than the information of available money.

LogSafari Summary Information

As we introduced in a previous section, we include an overall summary function intended to allow entry-level users to obtain some summary information about the log files. The summary information is output in HTML format, and a user can open it with any Web browser. In Figure 9, we show a screenshot of one such output.

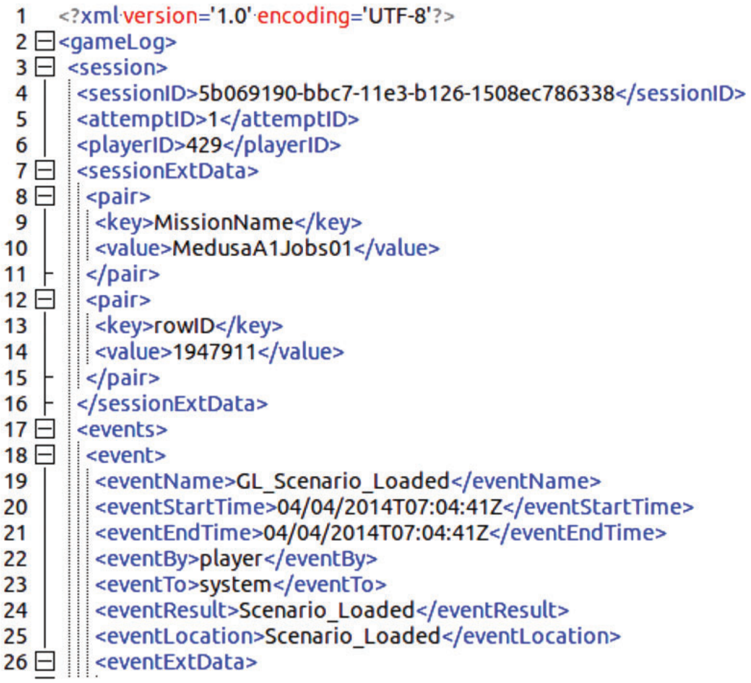
Summary and Discussion

In this report, we presented a systematic solution for initial stages of evidence identification from log files of G/SBAs. Our solution has two components: a well-structured data model and a suite of functionalities encapsulated into a Python package. We proposed a data model based on our investigation of various existing G/SBAs and implemented this data model in XML format. Moreover, we provided the corresponding XML schema for the data model with which all incoming log files are checked for compliance. Based on such a setup, we can ensure that the log files come in good shape, and we can reduce the pains of error handling when developing the analysis tools. On the basis of this data-structuring work, we developed a set of analysis functionalities in Python. As our research on G/SBAs moves ahead, the functionalities in the package will continue to become richer. We are planning a release of the package to users in the community in the near future.

Although the data model and the analysis tool go well with our existing logs, and are designed to be extensible based on our current best knowledge about the G/SBAs, we caution that they may not be adequate nor optimized for situations where the games/simulations are extremely complex or atypical. Readers and users are advised to use discretion when applying the data model to their specific G/SBAs.

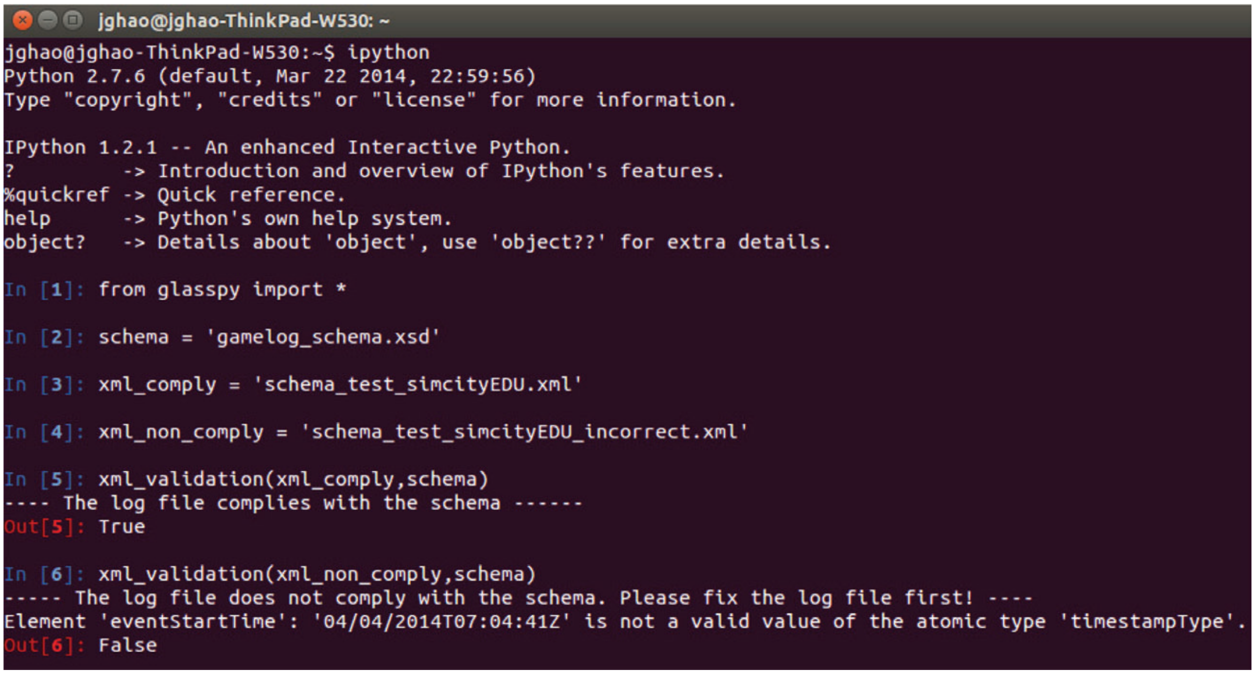
Acknowledgments

This research was supported in part by the ETS Foundational and Applied Statistics and Psychometrics Initiative (2014).



```
1   <?xml version='1.0' encoding='UTF-8'?>
2   <gameLog>
3    <session>
4     <sessionID>5b069190-bbc7-11e3-b126-1508ec786338</sessionID>
5     <attemptID>1</attemptID>
6     <playerID>429</playerID>
7     <sessionExtData>
8      <pair>
9       <key>MissionName</key>
10      <value>MedusaA1Jobs01</value>
11     </pair>
12     <pair>
13      <key>rowID</key>
14      <value>1947911</value>
15     </pair>
16    </sessionExtData>
17    <events>
18     <event>
19      <eventName>GL_Scenario_Loaded</eventName>
20      <eventStartTime>04/04/2014T07:04:41Z</eventStartTime>
21      <eventEndTime>04/04/2014T07:04:41Z</eventEndTime>
22      <eventBy>player</eventBy>
23      <eventTo>system</eventTo>
24      <eventResult>Scenario_Loaded</eventResult>
25      <eventLocation>Scenario_Loaded</eventLocation>
26      <eventExtData>
```

Figure 5 Snippet of schema_test_simcityEDU_incorrect.xml, which does not fully comply with our data model. The noncompliant part is the date format in Lines 20 and 21. All other parts comply with the data model.



```
jghao@jghao-ThinkPad-W530:~$ ipython
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
Type "copyright", "credits" or "license" for more information.

IPython 1.2.1 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: from glasspy import *

In [2]: schema = 'gamelog_schema.xsd'

In [3]: xml_comply = 'schema_test_simcityEDU.xml'

In [4]: xml_non_comply = 'schema_test_simcityEDU_incorrect.xml'

In [5]: xml_validation(xml_comply,schema)
---- The log file complies with the schema ------
Out[5]: True

In [6]: xml_validation(xml_non_comply,schema)
----- The log file does not comply with the schema. Please fix the log file first! ----
Element 'eventStartTime': '04/04/2014T07:04:41Z' is not a valid value of the atomic type 'timestampType'.
Out[6]: False
```

Figure 6 Example of extensible markup language schema validation using the function xml_validation in the glassPy package. When the log file complies with the schema, it will print out "The XML file complies with the schema," whereas "The XML file does not comply with the schema" as well as the reason why it does not comply will be displayed if the log file does not comply.

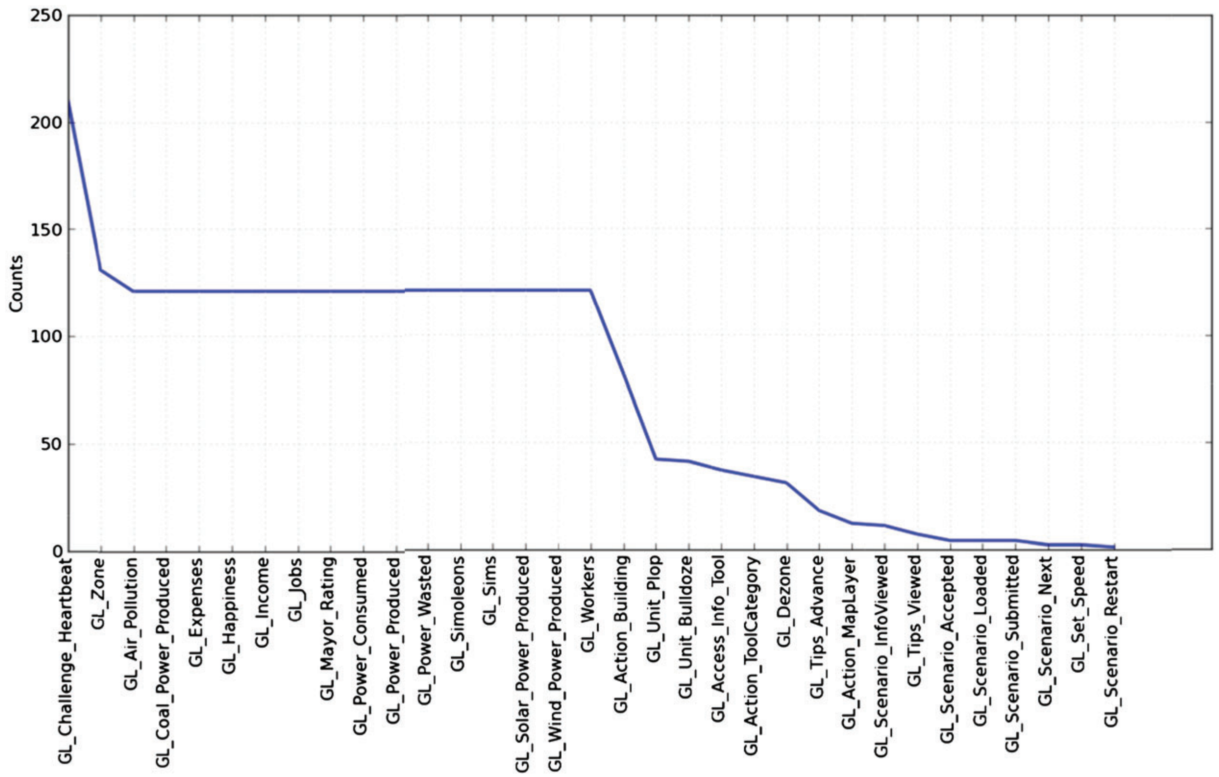


Figure 7 Frequency of events in one session of SimCityEDU. The *x* axis is for different game events, and the *y* axis is for the count (frequency) of the events.

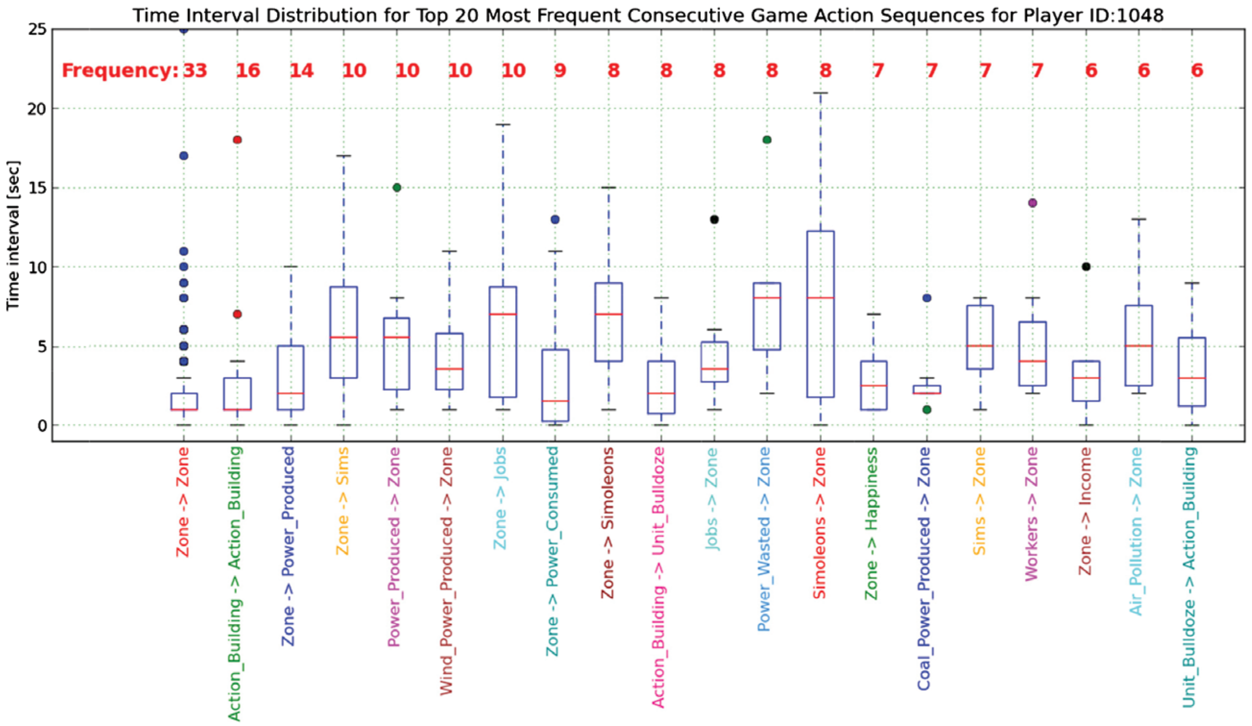
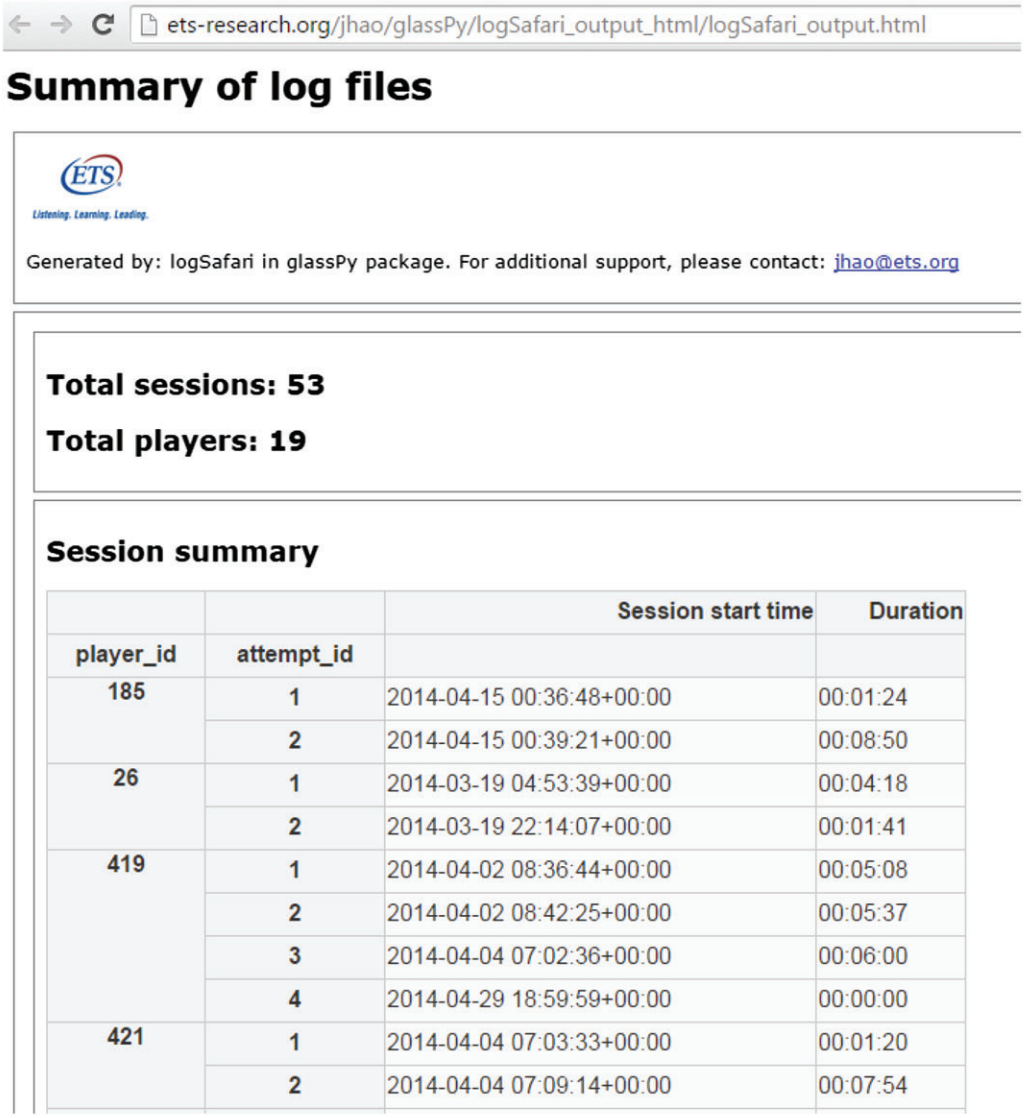


Figure 8 Distribution of the interevent time intervals for the most frequent event pairs (bigrams). The box plots show how the time intervals are distributed for different event pairs.



ets-research.org/jhao/glassPy/logSafari_output_html/logSafari_output.html

Summary of log files

Generated by: logSafari in glassPy package. For additional support, please contact: jhao@ets.org

Total sessions: 53

Total players: 19

Session summary

		Session start time	Duration
player_id	attempt_id		
185	1	2014-04-15 00:36:48+00:00	00:01:24
	2	2014-04-15 00:39:21+00:00	00:08:50
26	1	2014-03-19 04:53:39+00:00	00:04:18
	2	2014-03-19 22:14:07+00:00	00:01:41
419	1	2014-04-02 08:36:44+00:00	00:05:08
	2	2014-04-02 08:42:25+00:00	00:05:37
	3	2014-04-04 07:02:36+00:00	00:06:00
	4	2014-04-29 18:59:59+00:00	00:00:00
421	1	2014-04-04 07:03:33+00:00	00:01:20
	2	2014-04-04 07:09:14+00:00	00:07:54

Figure 9 Snippet of part of the output of the logSafari summary function.

Notes

[1] SimCityEDU was developed by GlassLab in collaboration with ETS and Pearson, with support from Electronic Arts, the Bill and Melinda Gates Foundation, and the John D. and Catherine T. MacArthur Foundation. GlassLab develops and researches next-generation learning games to support acquisition of critical 21st-century skills and creates tools to enable game developers to build better learning games and reach more learners.

[2] See http://www.w3.org/XML/

References

Bejar, I. I., Mattson, D., Wagner, M., Driscoll, G., & Hakkinen, M. T. (2014). *Accessibility, interoperability, and model-based test production: Leveraging the synergies* (Research Report No. RM-14-08). Princeton, NJ: Educational Testing Service.

Bergner, Y., Shu, Z., & von Davier, A. A. (2014). Visualization and confirmatory clustering of sequence data from a simulation-based assessment task. *Proceedings of the 7th International Conference on Educational Data Mining* (pp. 177–184). Retrieved from http://educationaldatamining.org/EDM2014/uploads/procs2014/long%20papers/177_EDM-2014-Full.pdf

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., & Yergeau, F. (1998). *Extensible markup language (XML)* (World Wide Web Consortium Recommendation No. REC-xml-19980210). Retrieved from http://www.w3.org/TR/1998/REC-xml-19980210

Gibson, D., & Jakl, P. (2015). Theoretical considerations for game-based e-learning analytics. In *Gamification in education and business* (Aufl. ed., pp. 403–416). Cham, Switzerland: Springer International.

Halverson, R., & Owen, V. E. (2014). Game-based assessment: An integrated model for capturing evidence of learning in play. *International Journal of Learning Technology*, *9*(2), 111–138.

Hao, J., & Kitchen, C. (2015). *Characterizing the time spending strategy in game based assessment: A hierarchical vectorization approach* (Unpublished manuscript).

Hao, J., Shu, Z., & von Davier, A. (2015). Analyzing process data from game/scenario-based tasks: An edit distance approach. *Journal of Educational Data Mining*, *7*(1), 33–50.

Hao, J., Smith, L., III, Mislevy, R., & von Davier, A. (2014). *Systems and methods for designing, parsing and mining of game log files*. U.S. patent 14/527,591.

Jurafsky, D., & Martin, J. H. (2000). *Speech and language processing*. Upper Saddle River, NJ: Prentice Hall.

Kerr, D., & Chung, G. K. (2012). Identifying key features of student performance in educational video games and simulations through cluster analysis. *Journal of Educational Data Mining, 4*(1), 144–182.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, *10*, 707–710.

McKinney, W. (2012). *Pandas: A Python data analysis library*. Retrieved from http://pandas.pydata.org/

Mislevy, R. J., Behrens, J. T., DiCerbo, K., & Levy, R. (2012). Design and discovery in educational assessment: Evidence centered design, psychometrics, and data mining. *Journal of Educational Data Mining*, *4*(1), 11–48.

Mislevy, R. J., & Haertel, G. D. (2006). Implications of evidence-centered design for educational testing. *Educational Measurement: Issues and Practice*, *25*(4), 6–20.

Mislevy, R. J., Oranje, A., Bauer, M. I., von Davier, A., Hao, J., Corrigan, S., … John, M. (2014). *Psychometric considerations in game-based assessment.* New York, NY: Institute for Play.

National Center for Education Statistics. (2013). *TEL WELL task*. Retrieved from http://nces.ed.gov/nationsreportcard/tel/wells_ite.maspx

van Rossum, G., Warsaw, B., & Coghlan, N. (2001). *PEP 8—Style guide for Python code*. Retrieved from http://www.python.org/devp/eps/pep-0008

von Davier, A., & Mislevy, R. (in press). Design and modeling frameworks for 21st century simulations and game-based assessments. In C. Wells & M. Faulkner-Bond (Eds.), *Educational measurement: From foundations to future*. New York, NY: Guilford Press.

Williams, R., Ochsenbein, F., Davenhall, C., Durand, D., Fernique, P., Giaretta, D., & Wicenec, A. (2002). *VOTable: A proposed XML format for astronomical tables*. Strasbourg, France: CDS.

Zhang, M., Hao, J., Li, C., & Deane, P. (in press). *Classification of writing patterns using keystroke log, Proceedings of the 80th Annual Meeting of Psychometric Society*.

Appendix

Game Log Schema

```
<?xml version="1.0"encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">

<!--
 Game Log Schema
 Version 1.0
 Authors:
  Lonnie Smith (lsmith@ets.org)
  Jiangang Hao (jhao@ets.org)

  Note: this is final version as of 3/16/2015

  (c) 2014, Educational Testing Service
-->

 <!--overall structure of the log-->

 <xs:element name="gameLog">
  <xs:complexType>
   <xs:sequence>
      <xs:element name="session" minOccurs="1" maxOccurs="unbounded">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="sessionID" type="idType" minOccurs="1" maxOccurs="1"/>
                   <xs:element name="playerID" type="idType" minOccurs="1" maxOccurs="1"/>
         <xs:element name="attemptID" type="idType" minOccurs="1" maxOccurs="1"/>
         <xs:element name="sessionExtData" type="dictType" minOccurs="0" maxOccurs="1"/>
         <xs:element name="eventSequence" minOccurs="1" maxOccurs="1">
          <xs:complexType>
           <xs:sequence>
            <xs:element name="event" type="eventType" minOccurs="1" maxOccurs="unbounded"/>
           </xs:sequence>
          </xs:complexType>
         </xs:element>
        </xs:sequence>
       </xs:complexType>
      </xs:element>


</xs:sequence>
  </xs:complexType>
 </xs:element>


 <!--Data type definitions -->


 <!—definition of idType. All identifiers must follow this rule -->
 <xs:simpleType name="idType">
  <xs:restriction base="xs:string">
   <xs:pattern value="([a-zA-Z0-9_,-:\-])*"/>
  </xs:restriction>
 </xs:simpleType>

 <!--definitionof timestampType. Follow subset of ISO 8601 standard, must use UTC -->
 <xs:simpleType name="timestampType">
  <xs:restriction base="xs:dateTime">
   <xs:pattern value="20\d{2}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}Z"/>
  </xs:restriction>
 </xs:simpleType>
```

```
  <!--definition of dictType for "extended" data (sequence of key/value pairs) -->
  <xs:complexType name="dictType">
   <xs:sequence>
    <xs:element name="pair" minOccurs="0" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="key" type="idType" minOccurs="1" maxOccurs="1"/>
       <xs:element name="value" type="xs:string" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>

  <!--definition of eventType-->
  <xs:complexType name="eventType">
   <xs:sequence>
    <xs:element name="eventName" type="idType" minOccurs="1" maxOccurs="1"/>
    <xs:element name="eventStartTime" type="timestampType" minOccurs="1" maxOccurs="1"/>
    <xs:element name="eventEndTime" type="timestampType" minOccurs="0" maxOccurs="1"/>
    <xs:element name="eventBy" type="idType" minOccurs="1" maxOccurs="1"/>
    <xs:element name="eventTo" type="idType" minOccurs="1" maxOccurs="1"/>
    <xs:element name="eventResult" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="eventLocation" type="idType" minOccurs="1" maxOccurs="1"/>
    <xs:element name="eventExtData" type="dictType" minOccurs="0"maxOccurs="1"/>
   </xs:sequence>
  </xs:complexType>

 </xs:schema>
```

Suggested citation:

Hao, J., Smith, L., Mislevy, R., von Davier, A., & Bauer, M. (2016). *Taming log files from game/simulation-based assessments: Data models and data analysis tools* (Research Report No. RR-16-10). Princeton, NJ: Educational Testing Service. http://dx.doi.org/10.1002/ets2.12096

Action Editor: Rebecca Zwick

Reviewers: Andreas Oranje and Diego Zapata Rivera