CRESST REPORT 790

Deirdre Kerr

Gregory K. W. K. Chung

Markus R. Iseli

THE FEASIBILITY OF USING CLUSTER ANALYSIS TO EXAMINE LOG DATA FROM EDUCATIONAL VIDEO GAMES

APRIL 2011

The National Center for Research on Evaluation, Standards, and Student Testing

Graduate School of Education & Information Sciences
UCLA | University of California, Los Angeles

The Feasibility of Using Cluster Analysis to Examine Log Data From Educational Video Games

CRESST Report 790

Deirdre Kerr, Gregory K. W. K. Chung, and Markus R. Iseli
CRESST/University of California, Los Angeles

April, 2011

National Center for Research on Evaluation,
Standards, and Student Testing (CRESST)
Center for the Study of Evaluation (CSE)
Graduate School of Education & Information Studies
University of California, Los Angeles
300 Charles E. Young Drive North
GSE&IS Bldg., Box 951522
Los Angeles, CA 90095-1522
(310) 206-1532

The work reported herein was supported under the Educational Research and Development Centers Program, PR/Award Number R305C080015.

To cite from this report, please use the following as your APA reference:
Kerr, D., Chung, G. K. W. K., & Iseli, M. R. (2011). *The feasibility of using cluster analysis to examine log data from educational video games* (CRESST Report 790). Los Angeles, CA: University of California, National Center for Research on Evaluation, Standards, and Student Testing (CRESST).

TABLE OF CONTENTS

THE FEASIBILITY OF USING CLUSTER ANALYSIS TO EXAMINE LOG DATA FROM EDUCATIONAL VIDEO GAMES

Deirdre Kerr, Gregory K. W. K Chung, and Markus R. Iseli
CRESST/University of California, Los Angeles

Abstract

Analyzing log data from educational video games has proven to be a challenging endeavor. In this paper, we examine the feasibility of using cluster analysis to extract information from the log files that is interpretable in both the context of the game and the context of the subject area. If cluster analysis can be used to identify patterns of thought as students play through the game, this method may be able to provide the information necessary to diagnose mathematical misconceptions or to provide targeted remediation or tailored instruction.

Introduction

One of the key issues for researchers examining the impact of educational video games is the analysis of the log data generated by these games. Without the ability to analyze these data, we may be able to determine whether or not students learned from a game but not precisely how or where this learning occurred. It is sometimes more important to know how a student plays a particular level of a game or solves a particular question on a test than it is to know the student's final score (Rahkila & Karjalainen, 1999). Log data can store complete student answers including strategies and mistakes (Merceron & Yacef, 2004), thereby letting the researcher record the learning behavior of students as they play the game (Romero & Ventura, 2007). In addition, the analysis of log data allows for the discovery, based on student usage data, of new knowledge about when and how learning occurs and what causes misunderstandings to arise within the game (Romero, Gonzalez, Ventura, del Jesus, & Herrera, 2009).

However, there are a number of reasons why educational researchers have not made a practice of analyzing log data beyond the extraction of basic summary statistics. Logs generate such large quantities of data that they can be very difficult to analyze (Romero et al., 2009). For instance, approximately 135 subjects playing a simple puzzle game for about half an hour can easily generate over 400,000 rows of log data (Chung et al., 2010). On top of the amount of information provided, the specific information gained from these usage statistics is not always easy to interpret (Romero & Ventura, 2007), as it can be very difficult to picture how student knowledge, learning, or misconceptions manifest themselves at the level of a specific event taken by the student in the course of the game. In addition, it can be very difficult to separate the noise from the substance given that log files are generally designed to

capture all student actions that have any chance of being relevant to learning, and it is not until after analysis that one would know which of those actions was actually relevant. Therefore, log files frequently contain large amounts of irrelevant data and require the use of both advanced statistical methods capable of dealing with large data sets and relevant background and domain knowledge to focus the analysis (Frawley, Piateski-Shapiro, & Matheus, 1992).

One promising method for analyzing log data is data mining. Data mining is a process that identifies frequent patterns in the data, despite the noise surrounding them, through the analysis of either general correlations or sequential correlations (Bonchi et al., 2001). Data mining summarizes or compresses the data set into a manageable number of variables that are nontrivial, implicit, previously unknown, and potentially useful (Frawley et al., 1992; Hand, Mannila, & Smyth, 2001). Though it has not yet found its way into the mainstream of educational research, data mining has been used regularly in fields such as engineering, chemistry, physics, astronomy, law enforcement, and publishing to identify key data in large data sets (Frawley et al., 1992).

There are four distinct families of data mining techniques: association rule mining that is used to find events that occur together, subgroup discovery that is used to identify interesting properties of subgroups, classification rule discovery that is used to identify defining characteristics of groups, and clustering that is used to discover patterns reflecting user behaviors (Romero et al., 2009). Clustering is a density estimation technique for identifying patterns within user actions that reflect differences in underlying attitudes, thought processes, or behaviors (Berkhin, 2006). It is particularly appropriate for the analysis of log data, as clustering is driven solely by the data at hand and is therefore ideal in instances in which little prior information is known (Jain, Murty, & Flynn, 1999).

Cluster analysis partitions actions into groups on the basis of a matrix of interobject similarities (James & McCulloch, 1990) by minimizing within-group distances compared to between-group distances so that actions classified as being in the same group are more similar to each other than they are to actions in other groups (Huang, 1998). Two actions will be considered similar by the cluster analysis if they are both performed by the same students. Actions will be considered different from each other if some students perform one of the actions and different students perform the other action. Properly used, cluster analysis algorithms can identify the latent dimensionality structure of a set of actions (Roussos, Stout, & Marden, 1998) to perform the necessary pattern reduction and simplification so that the patterns present in large data sets can be detected (Vogt & Nagel, 1992). In the case of log data from educational video games, the identified patterns, or clusters, would reflect the

different solution strategies and error patterns manifested by the students as they attempted to solve each game level.

In this report, we seek to determine to what extent cluster analysis can function as a valid tool for extracting non-trivial patterns that reflect underlying solution strategies or error patterns from educational video game log data. We will accomplish this by answering the following questions:

1. Will cluster analysis be able to pull clusters of actions from all levels, areas, or sections of a game, or only from certain specific ones?
2. Will cluster analysis consistently identify similar clusters in similar situations throughout the game?
3. Will the clusters that are identified be interpretable in the context of the game and subject area (i.e., will the actions in each cluster clearly reflect a specific underlying solution strategy or error pattern)?
4. What percentage of the data will be explained by the identified clusters?

Method

Study Design

The data used in this study come from the log files generated by an educational video game designed by CRESST and the University of Southern California's Game Innovation Lab to teach the addition of fractions (Chung et al., 2010). It has been our view throughout game development and testing that well-designed log files from a well-crafted educational video game would hold key information about how students conceptualized the underlying subject area. At all stages, we made design decisions reflecting this view.

Our game was specifically designed from the beginning so that the actions a student took would reflect the mathematical decisions being made and so that all key game mechanics were linked to mathematical operations (Chung et al., 2010). Similar care was taken in the design of the log files to ensure that actions we believed to reflect behaviors indicative of students' underlying mathematical understanding were intentionally logged. Additionally, the log files were designed so that the uniqueness of events was preserved and so that all key information was captured in detail (e.g., "an addition of coil of size *x* to coil of size *y* at position *z*," rather than "an addition of a coil"). Log data of this structure, created from a game utilizing this design strategy, are ideally suited for cluster analysis and will provide the best chance of uncovering new knowledge about how students conceptualize the problems they are asked to solve.

The game, called *Save Patch*, was designed to teach the addition of fractions. In this game, students are required to apply concepts underlying rational number addition to help the game character, Patch, bounce over obstacles to reach his home. To correctly solve each level, students must place trampolines at various locations along a one- or two-dimensional grid (see Figure 1). Students then drag coils onto the trampoline to make it bouncy. The distance Patch will bounce is the sum of all coil values added to the trampoline. For instance, if a student placed two 1/3 coils on a trampoline, Patch would bounce 2/3 of a unit.

In *Save Patch*, one whole unit is always the distance between two lines, and green dots indicate the size of the fractional pieces that should be used (see Figure 1). While any size coil can be placed on the trampoline initially, subsequent coils can only be added to the trampoline if they are the same size (i.e., have the same denominator).

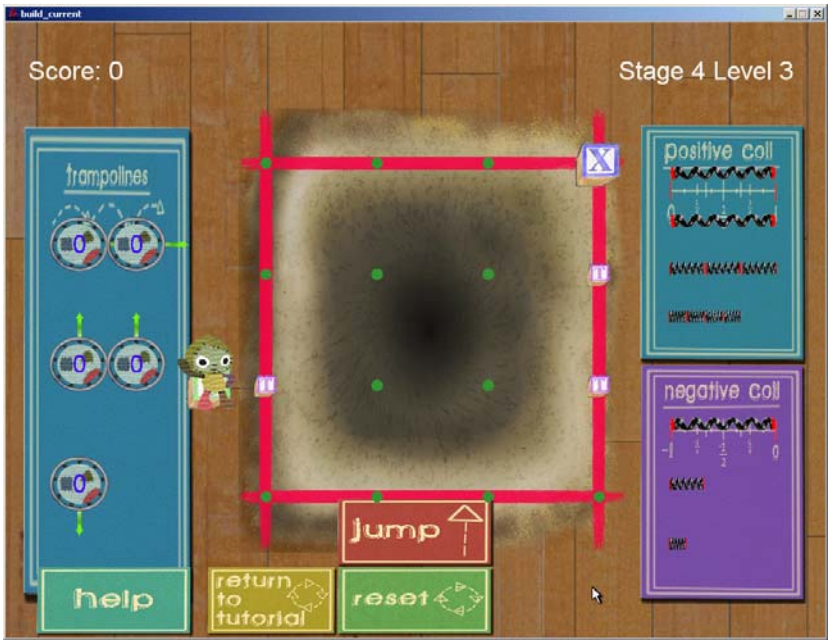


Figure 1. Screen shot of *Save Patch.*

As game play proceeds, trampolines must be placed at distances along the grid that are fractional parts of the whole unit. In early game levels, students are given the fractional unit coils. In later levels, students are shown how to break coils into fractional units. Because only coils that have identical units can be added together, students must be attentive to what the rational number means, what units are being added, what units are already on the trampoline, and how they will break coils into different size pieces. So, while students could add a coil that is 1/2 a unit to another coil that is 1/2 a unit, they are not allowed to add a coil that is 1/2 a unit to a coil that is a whole unit until the whole unit is broken into two 1/2-unit coils (i.e.,

2/2). At the time all three of these coils are added to the trampoline, the trampoline would show that it had 3/2 (rather than 1 1/2) units of bounce. This notation is intended to reinforce both the meaning of addition and the player's understanding of the meaning of rational numbers.

The sample of students who played *Save Patch* in this study includes 155 students (76 males and 79 females) from an urban school district in southern California. These students ranged from sixth to eighth grade and were in sixth grade math, Algebra Readiness, or Algebra 1 courses. All students played the game for approximately 40 minutes, and each action the students took in the game was logged automatically.

Cluster Analysis

To prepare the log data for cluster analysis, the data were separated into different data sets for each level, as correct actions in one level could be incorrect actions in another. The final data sets were formatted such that each row represented a student's attempt at completing the level, and each column represented an action that at least five students had taken when attempting to solve the level. These data sets were then imported into R for cluster analysis.

The clustering algorithm we chose to use in R is a fuzzy clustering algorithm called "fanny" (R Development Core Team, 2010). Fuzzy clustering, as opposed to hard clustering, allows an action to fall into multiple clusters if the action is part of more than one solution strategy or error pattern. We considered the potential overlap in clusters to be a very important aspect of the choice of algorithm, as certain actions may have been necessitated by game design regardless of which strategy the student was employing. Additionally, we used the Manhattan distance (rather than the Euclidean or Squared Euclidean distances commonly used) because our data were binary, indicating whether a particular student had performed a particular action in a given level. The Manhattan distance between two points (also known as the city block distance) is measured along axes at right angles only and is therefore appropriate for binary data, whereas the Euclidean distance is measured in a straight line between the two points. (See Appendix A for an elaboration on cluster analysis.)

Cluster analysis performed in this manner is intended to result in the identification of specific groups of actions within a level that correspond to underlying solution strategies or error patterns guiding student attempts to solve the level. Each action receives a score indicating its "percent belonging" to each cluster identified for the level. Scores close to 1 in a specific cluster and 0 in all other clusters indicate actions that serve to differentiate that cluster from other clusters, whereas actions with lower scores spread across multiple clusters

indicate poorly identified actions or actions representing steps used in multiple solution strategies or error patterns.

Solution Strategies

Cluster analysis of the game levels identified four legitimate solution strategies: the standard solution (identified in all levels), a solution using a larger denominator (identified in Stage 4 – Level 3 and Stage 5 – Level 1), an alternate legitimate solution (Stage 4 – Level 5; Stage 5 – Level 3; and Stage 6 – Levels 2, 3, and 4), or a solution using a shortcut (Stage 6 – Levels 2 and 3). In the standard solution, students solved the level in the manner the designers intended. The standard solution generally involved using the least common denominator of the given fractions and using the order in which the coils were presented to determine where to use whole units instead of fractional units. In Stage 6 – Level 2, that resulted in students placing 2/6 on Trampoline 1, 3/6 on Trampoline 2, and 1/6 on Trampoline 3. In the solution using a larger denominator, students solved the level with a denominator that was larger than the least common denominator. In Stage 6 – Level 2, that resulted in students placing 4/12 on Trampoline 1, 6/12 on Trampoline 2, and 2/12 on Trampoline 3. In the alternate legitimate solution, students either reversed the order of two equivalent pieces of the standard solution or simplified a part of the standard solution. In Stage 6 – Level 2, that resulted in students placing 1/3 on Trampoline 1, instead of 2/6. In the solution using a shortcut, students deliberately skipped one or more trampolines, getting to the other side in less than the maximum number of jumps allowed in that level. In Stage 6 – Level 2, that resulted in students placing 6/6 on Trampoline 1. Figure 2 shows a screen shot of this game level and the standard and alternate solutions.

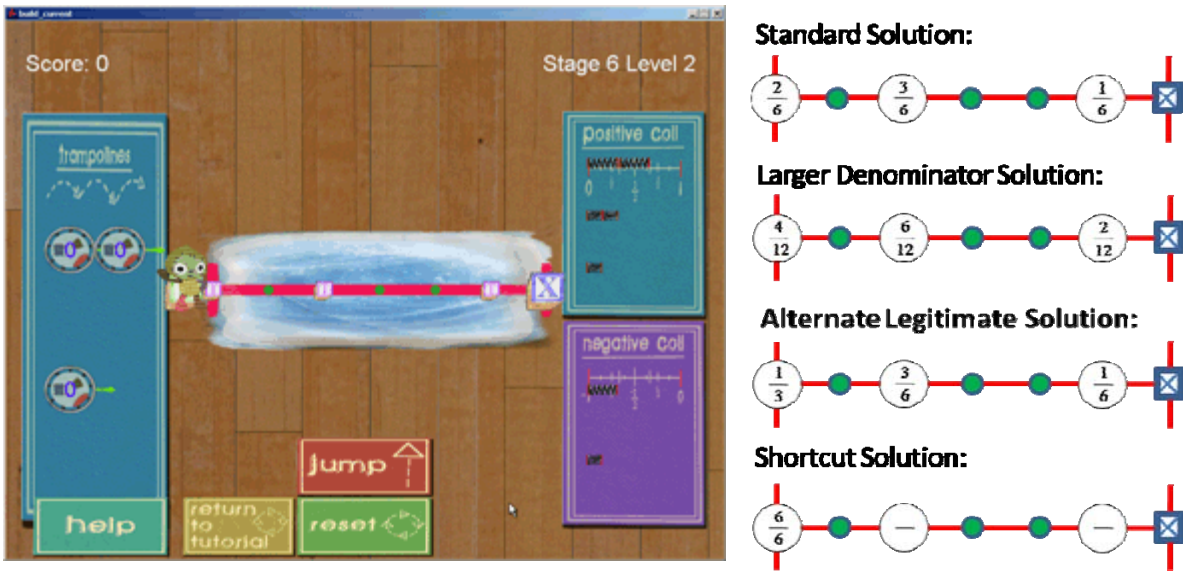


Figure 2. Stage 6 – Level 2 of *Save Patch*, illustrating different solution strategies.

Game Strategy Errors

The error patterns that were identified by the cluster analysis fell into two categories: errors involving game strategy and errors involving mathematical misconceptions. The errors involving game strategy included using all available resources in the order in which they were provided (Stage 3 – Levels 1 and 3; Stage 4 – Levels 3 and 4; Stage 5 – Level 1; and Stage 6 – Level 2) and the misuse of resources (Stage 2 – Level 1; Stage 3 – Level 3; Stage 4 – Levels 2 and 3; and Stage 6 – Levels 1 and 4). Students who used all resources in the order in which they were provided used this order to determine which fractions to place on which trampolines, rather than examining the grid to determine mathematically which fractions to place on which trampolines. In Stage 4 – Level 3, that resulted in students placing 2/1 on Trampoline 1, 3/3 on Trampoline 2, and 4/6 on Trampoline 3 (see Figure 3). Students who misused resources used the coils they were given in a manner that was technically correct but resulted in them running out of coils of the necessary size further on in the level. Generally, this misuse of resources involved using fractional units instead of a whole unit on a one-unit jump. In Stage 4 – Level 3, that resulted in students placing 3/3 on Trampoline 1 and then not having any 1/3 coils left for the second and third trampolines (see Figure 3). A misuse of resources is only clustered as an error if students are unable to solve the level this way. Using resources in a way different from what was intended when the level was designed but solving the level anyway would result in the identification of an alternate legitimate solution cluster.

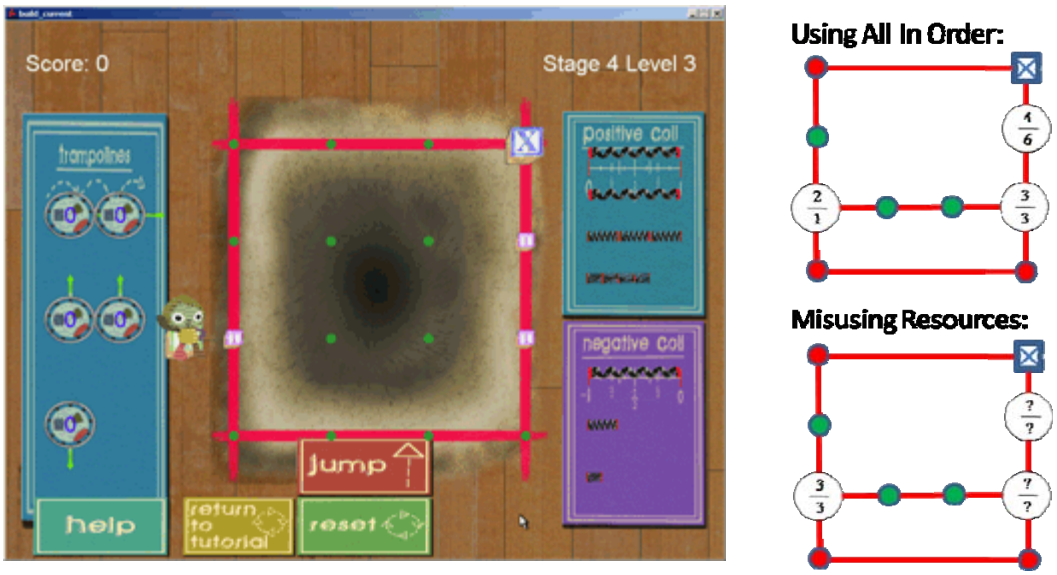


Figure 3. Stage 4 – Level 3 of *Save Patch*, illustrating errors involving game strategy.

Mathematical Errors

The errors involving mathematical misconceptions were largely naming errors, wherein the student was unable to correctly identify the denominator of the fraction that was to be used, rather than addition errors as we had anticipated. These error patterns identified by the cluster analysis indicated that students either made unitizing errors (Stage 2 – Level 1; Stage 4 – Levels 1 and 5; Stage 5 – Level 1; and Stage 6 – Levels 1 and 4), made partitioning errors involving counting points exclusively (Stage 3 – Level 2; Stage 4 – Level 4; Stage 5 – Levels 2 and 3; and Stage 6 – Level 3), made partitioning errors involving counting points inclusively (Stage 4 – Level 2; Stage 5 – Levels 2 and 3; and Stage 6 – Levels 3 and 4), or saw the solution as a mixed number (Stage 5 – Level 1). Students who made unitizing errors failed to pay attention to the red lines that indicated the length of a unit. Instead, such students appeared to see the entire grid as one unit. In Stage 5 – Level 1, that resulted in students placing 3/3 on Trampoline 1 (see Figure 4). Students who saw the solution as a mixed number tried to add a whole unit and a fractional unit without first converting the whole unit to the same denominator as the fractional unit. In Stage 5 – Level 1, that resulted in students trying to add 1/2 to 1/1 on Trampoline 1 (which the game does not recognize as a legitimate move), rather than converting 1/1 to 2/2 before adding 1/2 (see Figure 4).

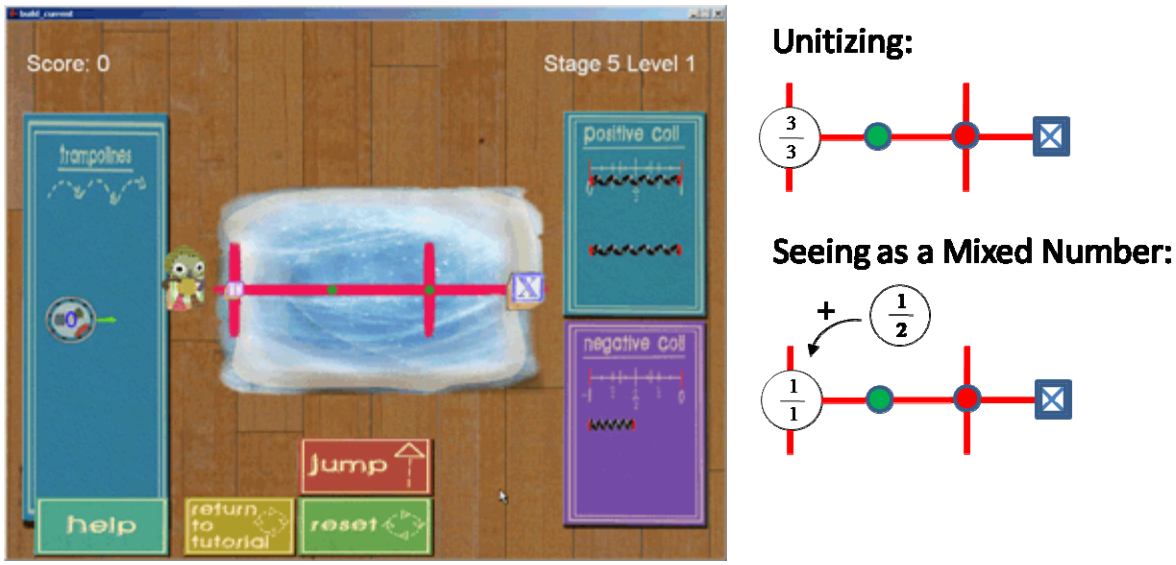


Figure 4. Stage 5 – Level 1 of *Save Patch*, illustrating unitizing errors and mixed number errors.

Students who made partitioning errors involving counting points exclusively appeared to be counting the points on the grid, rather than the spaces between the points, to determine the denominator of the fraction. In Stage 5 – Level 2, that resulted in students placing 2/2 on Trampoline 1, 2/2 on Trampoline 2, and 1/2 on Trampoline 3 (rather than solving the level in thirds). Students who made partitioning errors involving counting points inclusively apparently made the same error that students who counted points exclusively did, except that they also counted the points on the corners where the red lines intersected. In Stage 5 – Level 2, that resulted in students placing 2/4 on Trampoline 1, 2/4 on Trampoline 2, and 1/4 on Trampoline 3. See Figure 5 for a screen shot of this level and an illustration of the partitioning errors.

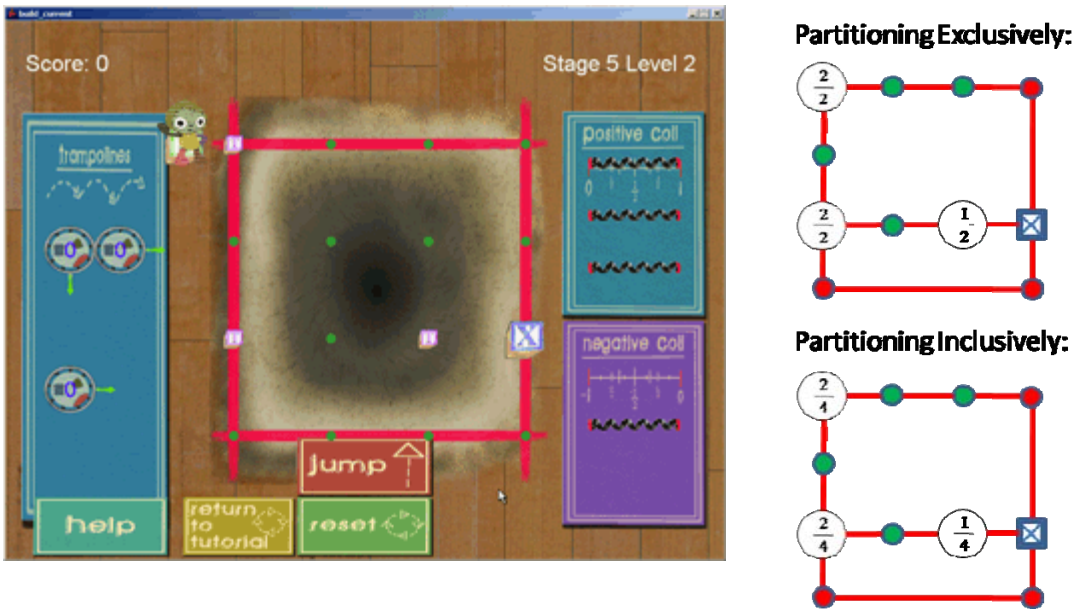


Figure 5. Stage 5 – Level 2 of *Save Patch*, illustrating partitioning errors.

Level Example

Solution strategies and error patterns were identified in each level by examining the actions that constituted each cluster. For example, in Stage 2 – Level 1 (see Figure 6), three clusters were identified: the standard solution, the misuse of resources error, and unitizing errors. The standard solution was identified because the cluster included the following actions: placing 1/1 on Trampoline 1, placing 1/2 on Trampoline 2, and placing 1/2 on Trampoline 3. These are the actions that would be expected of a student who knew the answer to the problem. The misuse of resources error was identified because the cluster included the following actions: placing 1/2 on Trampoline 1, and placing a second 1/2 on Trampoline 1. These actions are consistent with what a student would do if they used both available 1/2 coils on the wrong trampoline and then didn't have any 1/2 coils left to use on Trampoline 2 or Trampoline 3. These actions are only identified as being an error because the student runs out of the necessary resources; if there were additional 1/2 coils available, these actions would result in a correct solution. The unitizing errors were identified because the cluster included the following actions: placing 1/4 on Trampoline 2 and placing 1/4 on Trampoline 3. These actions are consistent with what a student would do if they ignored the red lines and thought that the whole screen was one unit. In that case, they would see the unit as being broken up into four pieces, with the first jump representing two of those pieces, and the second and third jumps each representing one of those pieces.

Appendix B lists, for each level, all of the actions identified by the cluster analysis as belonging to each cluster, and Appendix C and Appendix D list the SPSS and R codes used to run the analysis.

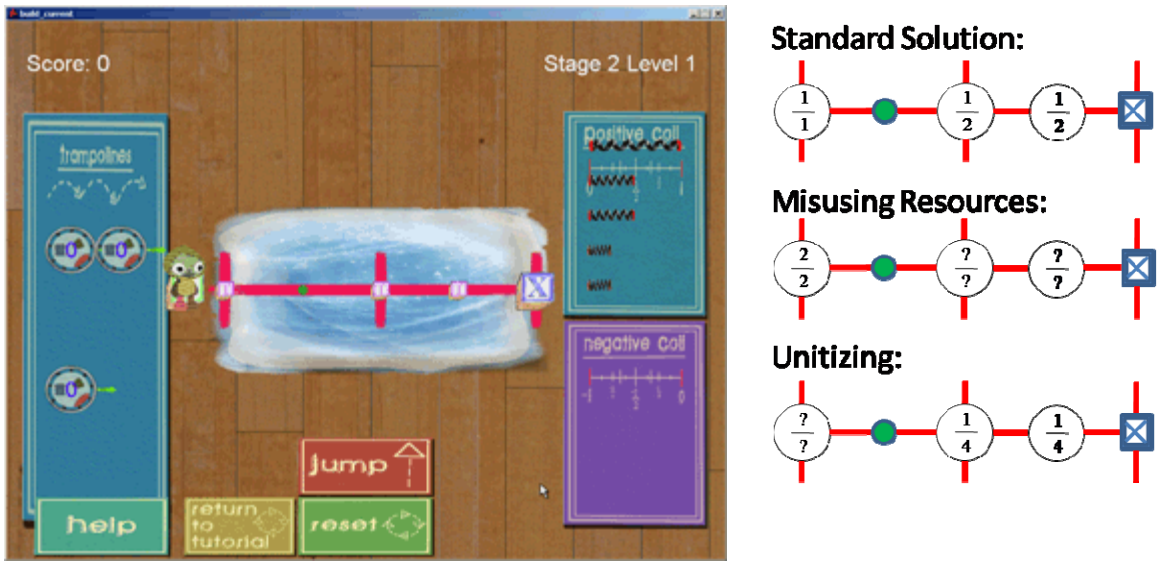


Figure 6. Stage 2 – Level 1 of *Save Patch*, illustrating the identified clusters for the level.

Results

Before changing instruction or providing remediation based on the error patterns identified in the previous analysis, it is important to examine their validity. Because cluster analysis is designed to uncover the latent dimensionality of data sets that are so large that this structure is obscured, the interpretation of the validity of the identified clusters is inevitably application-dependent and somewhat subjective (Hand et al., 2001). However, some basic questions regarding the validity of the analysis can still be answered, namely:

1. Did cluster analysis pull clusters of actions from all levels of the game?
2. Did cluster analysis consistently identify similar clusters in similar situations throughout the game?
3. Were the clusters that we identified interpretable in the context of the game and subject area (i.e., did the actions in each cluster clearly reflect a specific underlying solution strategy or error pattern)?
4. What percentage of the data was explained by the identified clusters?

To answer these questions, we tracked each cluster as it was identified, compiling information about the frequency of the occurrence of the cluster across the 16 levels as well as the average percentage of attempts the students made to solve the levels that fell in that

cluster (see Table 1). An attempt at solving a level is defined as a series of actions taken by a student in a given level that culminate in a death, a reset, or a successful solution. The average percentage of attempts was calculated only for levels in which the cluster was identified. The only two levels where no clusters were identified (Stage 1 – Level 1 and Stage 2 – Level 2) were close to the beginning of the game and were easy enough (i.e., students were given one coil and one trampoline and had to jump one whole space) that there was virtually no variance in the way students played them. All other levels contained at least one cluster reflecting a legitimate solution strategy, at least one cluster reflecting a specific type of error, and an additional cluster of unknown errors (see Appendix E). These results indicate that cluster analysis can pull clusters from all game levels, provided there is enough variance in student performance that multiple solution strategies or error patterns were actually employed in the level.

Each of the identified clusters is listed in Table 1. The standard solution and unexplained errors were found in all levels of the game. While no other cluster appeared in all levels, each of the other clusters appeared in a majority of the levels where that particular error or solution strategy was allowable by level design, except for the solution using a larger denominator, which was identified in only two of the levels and in only 6% of the attempts made on those levels.

The alternate legitimate solution was identified in five of the seven levels in which it was possible and accounted for an average of 26% of the attempts on those five levels. The solution using a shortcut was identified in only three of the 10 levels in which it was possible and accounted for an average of 8% of the attempts on those levels. The error pattern using all resources in the order in which they were provided was identified in six out of nine levels in which it was possible and accounted for an average of 19% of the attempts on those levels. The error pattern involving the misuse of resources was identified in six out of seven levels in which it was possible and accounted for an average of 11% of the attempts on those levels. The unitizing errors were identified in six out of eight levels in which they were possible and accounted for an average of 25% of the attempts on those levels. The errors involving partitioning exclusively were identified in five out of nine levels in which they were possible and accounted for an average of 39% of the attempts on those levels. The errors involving partitioning inclusively were identified in five out of 10 levels in which they were possible and accounted for an average of 9% of the attempts on those levels. The error involving seeing the solution as a mixed number was identified in one out of two levels in which it was possible and accounted for 22% of the attempts in that level.

Table 1

Solution Strategies and Error Patterns Identified by Cluster Analysis

Cluster identified	Identified frequency[a]	Percentage of attempts[b]
Solution strategies		
Standard solution	16	26.5%
Solution using a larger denominator	2	6.2%
Alternate legitimate solution	5	26.0%
Solution using a shortcut	3	8.4%
Error patterns involving game strategy		
Using all resources in the order they are provided	6	19.1%
Misuse of resources	6	10.9%
Error patterns involving mathematical misconceptions		
Unitizing errors	6	23.6%
Partitioning errors involving counting points exclusively	5	37.0%
Partitioning errors involving counting points inclusively	5	9.1%
Seeing the solution as a mixed number	1	22.4%
Unexplained errors	16	26.4%

[a]Identified frequency is the number of levels (out of 16) in which the cluster was identified.
[b]Percentage of attempts is calculated only for levels in which the cluster was identified.

The identification of errors across levels indicates that these error patterns are not specific to the level in which they were identified, but rather surfaced periodically throughout the game when allowed by level design (see Appendix E for a list of which clusters were possible in which levels). That only four different types of legitimate solution strategies and six different types of error patterns were identified, and that these clusters were extracted from a variety of different levels rather than each level having its own unique set of errors, show that cluster analysis did consistently identify similar clusters in similar situations.

The results of the cluster analysis were also easily interpretable. In all cases, the actions identified as belonging to a given cluster were interpretable in either the context of the game (in the case of error patterns involving game strategy) or in the context of the subject area (in the case of errors involving mathematical misconceptions). Additionally, the percentage of student attempts to solve a level that remained in the unexplained error cluster, and were therefore essentially unidentified by the cluster analysis, averaged 26.4% across all levels and ranged from 11.4% to 38.5%. These results indicate that the cluster analysis was able to account for an average of 73.6% of the attempts taken by students in each level.

Discussion

The overall performance of cluster analysis in this study indicates its potential for researchers interested in analyzing educational video games and simulations. Because the cluster analysis was able to pull clusters from nearly all levels of the game, and those clusters accounted for 73.6% of attempts made by students on those levels, the analysis clearly provides enough data to be worthwhile. Additionally, its ability to consistently identify similar clusters in similar situations suggests that cluster analysis may be a reliable way of examining log data. Most important, however, is that cluster analysis identified error patterns that clarified both the mathematical misconceptions students were employing and the game strategies that students utilized to solve the levels.

The performance of cluster analysis in this study indicates that it could be a valid tool for analyzing complex problem solving situations in educational video games or simulations. Because it pulls out patterns of actions (and therefore patterns of thought) that are not visible through either direct examination or standard summary statistics, cluster analysis provides a means of synthesizing log data that is difficult to do otherwise. Once the actions taken in the game are distilled into clusters of behaviors that reflect cognition, these clusters hold great potential for diagnosing mathematical misconceptions in students and for providing remediation and/or tailored instruction through video games or simulations. Provided that additional validity evidence can be collected to help determine how well the clusters reflect our interpretation of them, cluster analysis may well become a valuable tool for designing games, curriculum, or remediation techniques that result in higher learning outcomes due to their ability to focus on particular patterns of behavior.

References

Berkhin, R. (2006). A survey of clustering data mining techniques. In J. Kogan, C. Nicholas, & M. Teboulle (Eds.), *Grouping multidimensional data* (pp. 25–72). New York, NY: Springer.

Bonchi, F., Giannoti, F., Gozzi, C., Manco, G., Nanni, M., Pedreschi, D., ... Ruggieri, S. (2001). Web log data warehouses and mining for intelligent web caching. *Data & Knowledge Engineering, 39,* 165–189. doi:10.1016/S0169-023X(01)00038-6

Chung, G. K. W. K., Baker, E. L., Vendlinski, T. P., Buschang, R. E., Delacruz, G. C., Michiuye, J. K., & Bittick, S. J. (2010, April). Testing instructional design variations in a prototype math game. In R. Atkinson (Chair), *Current perspectives from three national R&D centers focused on game-based learning: Issues in learning, instruction, assessment, and game design*. Structured poster session at the annual meeting of the American Educational Research Association, Denver, CO.

Frawley, W. J., Piateski-Shapiro, G., & Matheus, C. J. (1992). Knowledge discovery in databases: An overview. *AI Magazine, 13*(3), 57–70. doi:10.1007/3540635149_30

Hand, D., Mannila, H., & Smyth, P. (2001). *Principles of data mining*. Cambridge, MA: MIT Press.

Huang, Z. (1998). Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data Mining and Knowledge Discovery, 2,* 283–304. doi:10.1023/A:1009769707641

Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data clustering: A review. *ACM Computing Surveys, 31,* 264–323. doi:10.1145/331499.331504

James, F., & McCulloch, C. (1990). Multivariate analysis in ecology and systematic: Panacea or Pandora's box? *Annual Review of Ecology and Systematics, 21,* 129–166. doi:10.1146/annurev.es.21.110190.001021

Merceron, A., & Yacef, K. (2004). Mining student data captured from a web-based tutoring tool: Initial exploration and results. *Journal of Interactive Learning Research, 15,* 319–346.

R Development Core Team. (2010). R: A Language and Environment for Statistical Computing [Computer software]. Retrieved from http://www.R-project.org

Rahkila, M., & Karjalainen, M. (1999). Evaluation of learning in computer based education using log systems. *Proceedings of 29th ASEE/IEEE Frontiers in Education Conference (FIE '99),* 16–22.

Romero, C., Gonzalez, P., Ventura, S., del Jesus, M. J., & Herrera, F. (2009). Evolutionary algorithms for subgroup discovery in e-learning: A practical application using Moodle data. *Expert Systems With Applications, 39,* 1632–1644. doi:10.1016/j.eswa.2007.11.026

Romero, C., & Ventura, S. (2007). Educational data mining: A survey from 1995 to 2005. *Expert Systems with Applications, 35,* 135–146. doi:10.1016/j.eswa.2006.04.005

Roussos, L., Stout, W., & Marden, J. (1998). Using new proximity measures with hierarchical cluster analysis to detect multidimensionality. *Journal of Educational Measurement, 35*, 1–30. doi:10.1111/j.1745-3984.1998.tb00525.x

Vogt, W., & Nagel, D. (1992). Cluster analysis in diagnosis. *Clinical Chemistry*, *38*, 182–198.

Appendix A:
Cluster Analysis Basics

Cluster analysis uses a distance metric. The distances between each of the items are calculated and plotted in *n*-dimensional space, where *n* is the number of items to be clustered (see Figure A1 for an example). In only two or three dimensions, the human eye and brain can determine clusters easily just by visual examination. Things that are close to each other and far from other groups belong to a cluster. Cluster analysis seeks to perform this same process mathematically when the number of dimensions exceeds the ability of human visual interpretation.

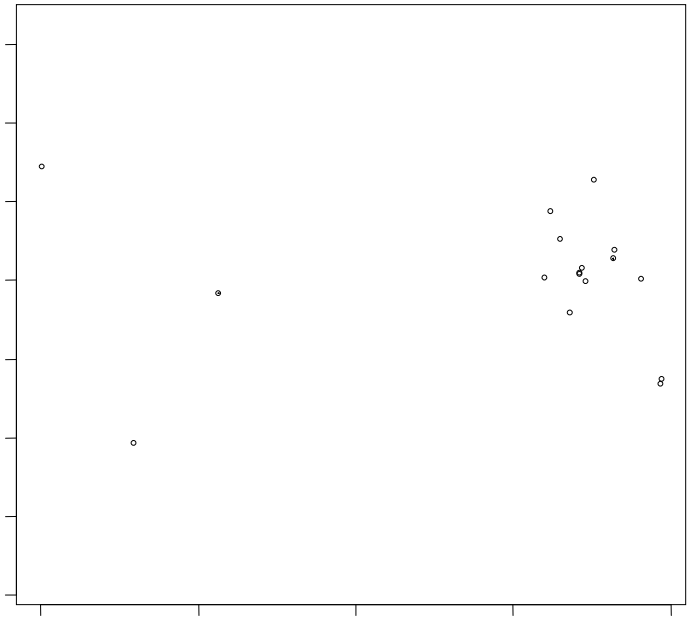


Figure A1. Items plotted in *n*-dimensional space.

To do this, a cluster analysis algorithm that seeks to partition the items into clusters generally begins by choosing *p* points at random, where *p* is the number of clusters desired (see Figure A2). Then, the algorithm checks the distances from each point to the initial *p* points to determine which of those points it is closest to. As the algorithm iterates, the initial points may be abandoned and the *p* points may shift to points closer to the center of the cluster.

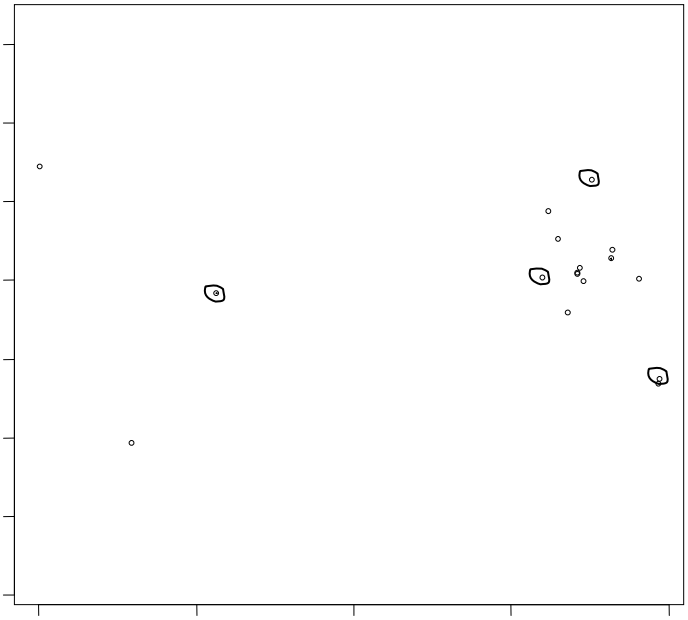


Figure A2. Initial *p* points chosen for cluster analysis.

Membership in a cluster is then determined by minimizing the ratio between the sum of the distances between a given point and all other points in the same cluster and the sum of distances between that point and all points outside of the cluster. This results in clusters that are relatively compact and well separated from each other, such as can be seen in Figure A3.

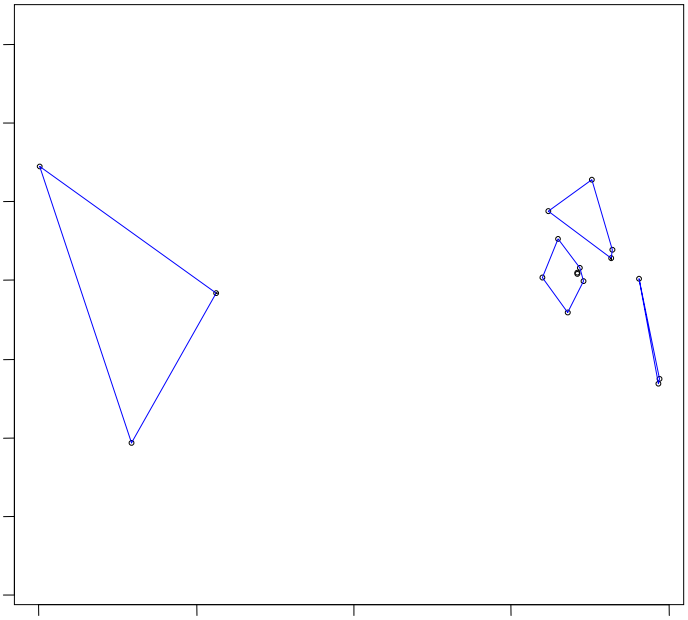


Figure A3. Final clusters identified by cluster analysis.

Appendix B:
Extracted Clusters by Level

This appendix lists the actions identified through cluster analysis as belonging to each cluster in each level. Trampolines are numbered (T1, T2, etc.) in the order in which they are bounced on in the level. OTHER refers to actions that were not recorded in the log file as taking place on a particular trampoline. The mnemonic codes generated from the log data to name the individual actions in the clusters are listed below the short descriptions of these actions inside the cells of the table.

Table B1

Stage 2 – Level 1 Clusters

	Trampoline		
Cluster	T1	T2	T3
S1: Standard Solution	1/1 on T1 ACRT_POS1.0_COIL1.1_YIELD1.1	1/2 on T2 ACRT_POS3.0_COIL1.2_YIELD1.2	1/2 on T3 ACRT_POS4.0_COIL1.2_YIELD1.2
E2: Misusing Resources	1/2 on T1 2/2 on T1 ACRT_POS1.0_COIL1.2_YIELD1.2 ACRT_POS1.0_COIL1.2_YIELD2.2		
E3: Unitizing Errors		1/4 on T2 ACRT_POS3.0_COIL1.4_YIELD1.4	1/4 on T3 ACRT_POS4.0_COIL1.4_YIELD1.4

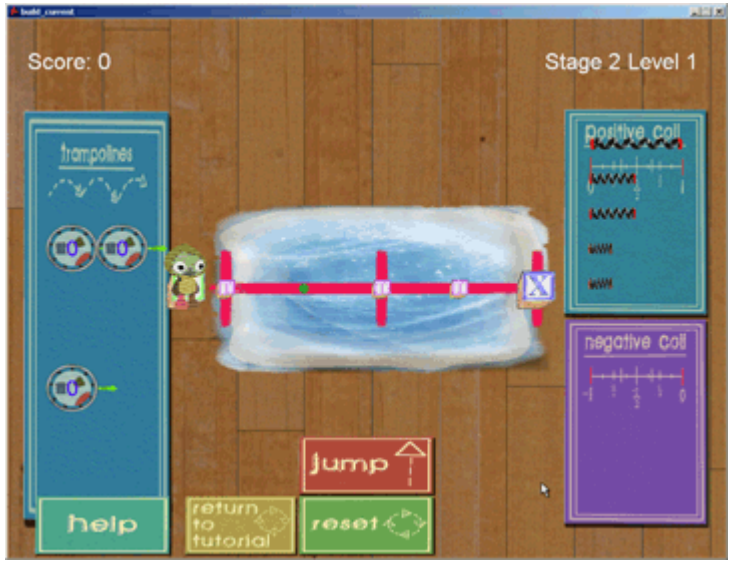


Table B2

Stage 3 – Level 1 Clusters

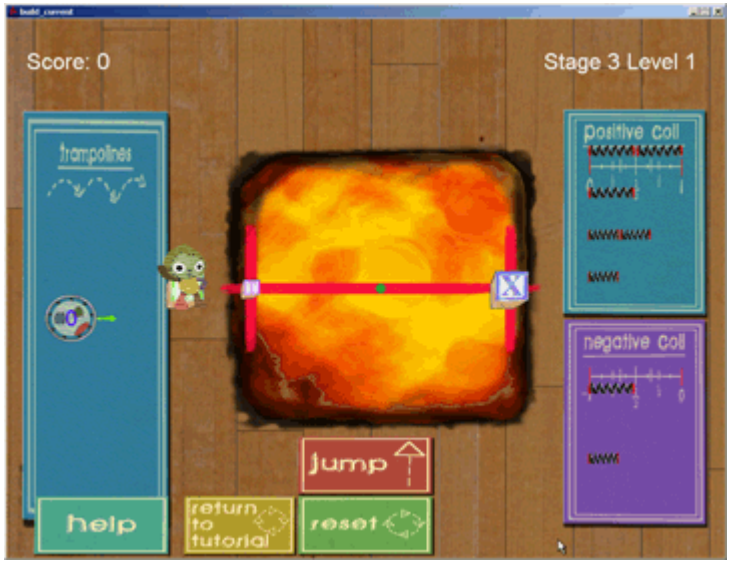
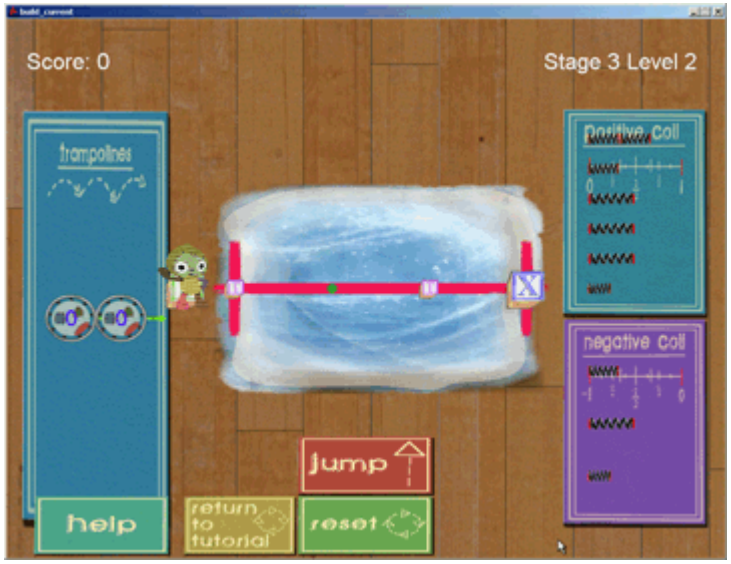
	Trampoline	
Cluster	T1	OTHER
S1: Standard Solution	1/2 on T1 2/2 on T1 ACRT_POS1.0_COIL1.2_YIELD1.2 ACRT_POS1.0_COIL1.2_YIELD2.2	
E1: All in Order	3/2 on T1 ACRT_POS1.0_COIL1.2_YIELD3.2	ADD 1/3 to 1/2 ACWR_COIL1.3_TRAMP1.2
		

Table B3

Stage 3 – Level 2 Clusters

Cluster	Trampoline		
	T1	T2	OTHER
S1: Standard Solution	1/3 on T1 2/3 on T1 ACRT_POS1.0_COIL1.3_YIELD1.3 ACRT_POS1.0_COIL1.3_YIELD2.3	1/3 on T2 ACRT_POS3.0_COIL1.3_YIELD1.3	
E4: Partitioning Exclusively	1/2 on T1 2/2 on T1 ACRT_POS1.0_COIL1.3_YIELD1.3 ACRT_POS1.0_COIL1.3_YIELD2.3	1/2 on T2 1/4 on T2[1] ACRT_POS3.0_COIL1.3_YIELD1.3 ACRT_POS3.0_COIL1.3_YIELD1.3	Add 1/3 to 1/2[1] ACWR_COIL1.3_TRAMP1.2



[1]These actions were identified as belonging to this cluster by the analysis but do not appear to belong to this error pattern.

Table B4

Stage 3 – Level 3 Clusters

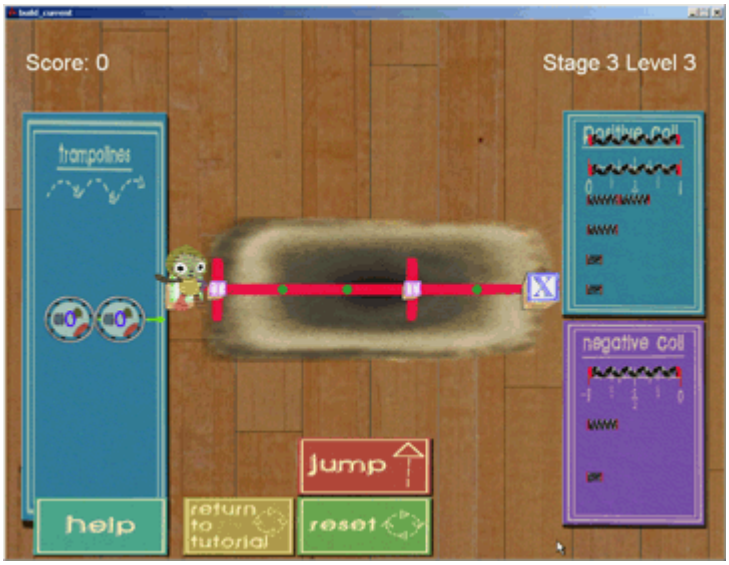
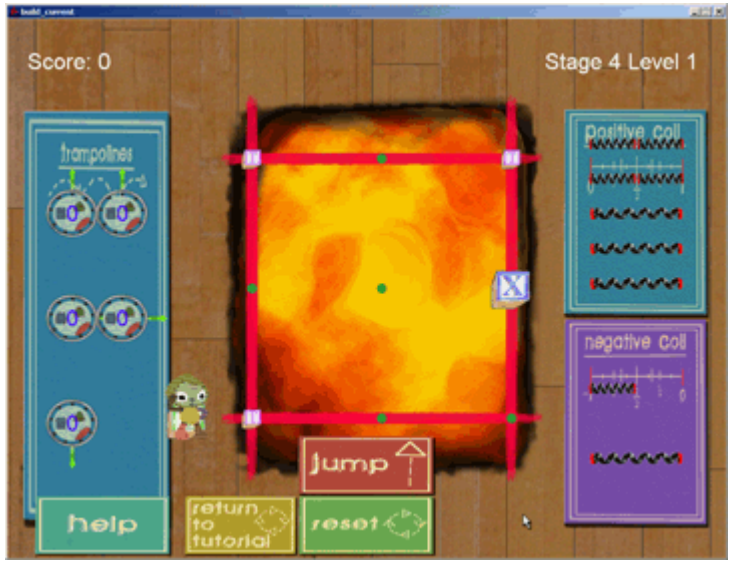
	Trampoline	
Cluster	T1	T2
S1: Standard Solution	1/1 on T1 ACRT_POS1.0_COIL1.1_YIELD1.1	1/3 on T2 2/3 on T2 ACRT_POS4.0_COIL1.3_YIELD1.3 ACRT_POS4.0_COIL1.3_YIELD2.3
E1: All in Order	2/1 on T1 ACRT_POS1.0_COIL1.1_YIELD2.1	1/6 on T2 2/6 on T2 ACRT_POS4.0_COIL1.6_YIELD1.6 ACRT_POS4.0_COIL1.6_YIELD2.6
E2: Misusing Resources	1/3 on T1 2/3 on T1 3/3 on T1 ACRT_POS1.0_COIL1.3_YIELD1.3 ACRT_POS1.0_COIL1.3_YIELD2.3 ACRT_POS1.0_COIL1.3_YIELD3.3	
		

Table B5

Stage 4 – Level 1 Clusters

Cluster	Trampoline			
	T1	T2	T3	OTHER
S1: Standard Solution	1/1 on T1 ACRT_POS1.2_COIL1.1_YIELD1.1	1/1 on T2 ACRT_POS1.0_COIL1.1_YIELD1.1	1/2 on T3 ACRT_POS3.0_COIL1.2_YIELD1.2	
E3: Unitizing Errors	2/1 on T1 ACRT_POS1.2_COIL1.1_YIELD2.1	1/2 on T2 2/2 on T2 3/2 on T2 4/2 on T2 ACRT_POS1.0_COIL1.2_YIELD1.2 ACRT_POS1.0_COIL1.2_YIELD2.2 ACRT_POS1.0_COIL1.2_YIELD3.2 ACRT_POS1.0_COIL1.2_YIELD4.2	1/1 on T3 ACRT_POS3.0_COIL1.1_YIELD1.1	Add 1/2 to 1/1[1] ACWR_COIL1.2_TRAMP1.1



[1] These actions were identified as belonging to this cluster by the analysis but do not appear to belong to this error pattern.

Table B6

Stage 4 – Level 2 Clusters

	Trampoline				
Cluster	T1	T2	T3	T4	T5
S1: Standard Solution	1/1 on T1 ACRT_POS1.0_COIL1.1_YIELD1.1	1/1 on T2 ACRT_POS4.0_COIL1.1_YIELD1.1	1/3 on T3 ACRT_POS4.3_COIL1.3_YIELD1.3	1/3 on T4 ACRT_POS5.3_ COIL1.3_YIELD1.3	1/3 on T5 ACRT_POS6.3_ COIL1.3_YIELD1.3
S4: Shortcut Solution			2/3 on T3 3/3 on T3 ACRT_POS4.3_COIL1.3_YIELD2.3 ACRT_POS4.3_COIL1.3_YIELD3.3		
E2: Misusing Resources	1/3 on T1 2/3 on T1 3/3 on T1 ACRT_POS1.0_COIL1.3_YIELD1.3 ACRT_POS1.0_COIL1.3_YIELD2.3 ACRT_POS1.0_COIL1.3_YIELD3.3	1/3 on T2 2/3 on T2 3/3 on T2 ACRT_POS4.0_COIL1.3_YIELD1.3 ACRT_POS4.0_COIL1.3_YIELD2.3 ACRT_POS4.0_COIL1.3_YIELD3.3			
E5: Partitioning Inclusively	1/4 on T1 2/4 on T1 3/4 on T1 ACRT_POS1.0_COIL1.4_YIELD1.4 ACRT_POS1.0_COIL1.4_YIELD2.4 ACRT_POS1.0_COIL1.4_YIELD3.4	1/4 on T2 2/4 on T2 3/4 on T2 ACRT_POS4.0_COIL1.4_YIELD1.4 ACRT_POS4.0_COIL1.4_YIELD2.4 ACRT_POS4.0_COIL1.4_YIELD3.4	1/4 on T3 ACRT_POS4.3_COIL1.4_YIELD1.4	1/4 on T4 ACRT_POS5.3_ COIL1.4_YIELD1.4	1/4 on T4 ACRT_POS6.3_ COIL1.4_YIELD1.4

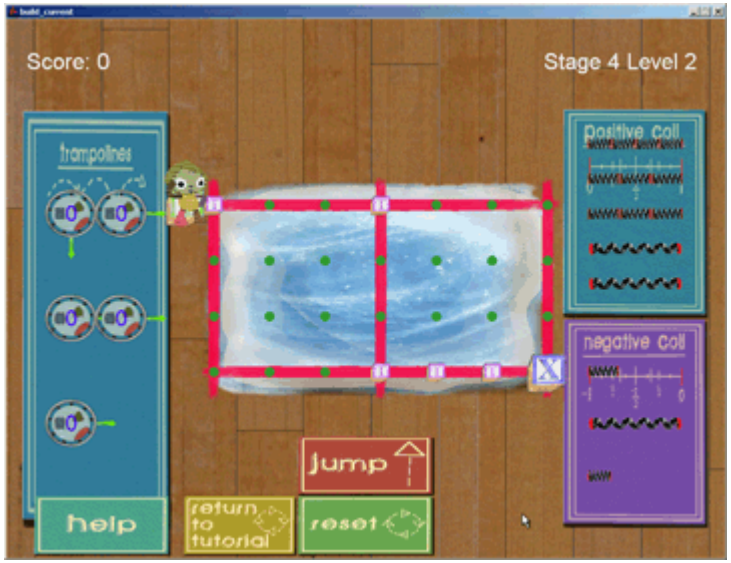

Score: 0
Stage 4 Level 2
trampolines
Positive Coil
negative Coil
Jump
help
return to tutorial
reset

Table B7

Stage 4 – Level 3 Clusters

	Trampoline		
Cluster	T1	T2	T3
S1: Standard Solution	1/1 on T1 ACRT_POS1.2_COIL1.1_YIELD1.1	1/3 on T2 ACRT_POS4.2_COIL1.3_YIELD1.3	1/3 on T3 ACRT_POS4.1_COIL1.3_YIELD1.3
S3: Larger Denominator		1/6 on T2 2/6 on T2 ACRT_POS4.2_COIL1.6_YIELD1.6 ACRT_POS4.2_COIL1.6_YIELD2.6	
E1: All in Order	2/1 on T1 ACRT_POS1.2_COIL1.1_YIELD2.1	2/3 on T2 ACRT_POS4.2_COIL1.3_YIELD2.3	1/6 on T3 2/6 on T3 3/6 on T3 4/6 on T3 ACRT_POS4.1_COIL1.6_YIELD1.6 ACRT_POS4.1_COIL1.6_YIELD2.6 ACRT_POS4.1_COIL1.6_YIELD3.6 ACRT_POS4.1_COIL1.6_YIELD4.6
E2: Misusing Resources	1/3 on T1 2/3 on T1 3/3 on T1 ACRT_POS1.2_COIL1.1_YIELD1.3 ACRT_POS1.2_COIL1.1_YIELD2.3 ACRT_POS1.2_COIL1.1_YIELD3.3		

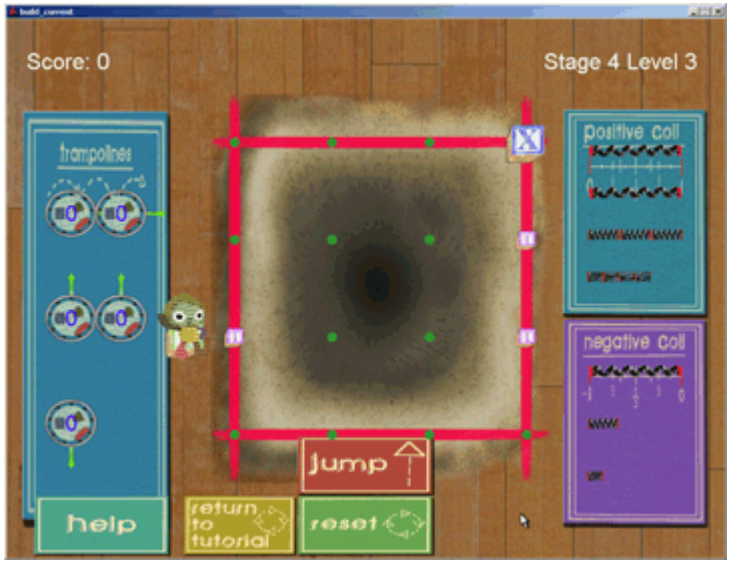

Score: 0
Stage 4 Level 3
trampolines
positive coil
negative coil
Jump
help
return to tutorial
reset

Table B8

Stage 4 – Level 4 Clusters

Cluster	Trampoline			
	T1	T2	T3	T4
S1: Standard Solution	1/3 on T1 2/3 on T1 ACRT_POS1.0_COIL1.3_YIELD1.3 ACRT_POS1.0_COIL1.3_YIELD2.3	1/3 on T2 ACRT_POS1.2_COIL1.3_YIELD1.3	1/3 on T3 2/3 on T3 ACRT_POS2.2_COIL1.3_YIELD1.3 ACRT_POS2.2_COIL1.3_YIELD2.3	1/3 on T4 ACRT_POS4.2_COIL1.3_YIELD1.3
E1: All in Order	1/2 on T1 ACRT_POS1.0_COIL1.2_YIELD1.2		1/2 on T3 ACRT_POS2.2_COIL1.2_YIELD1.2	
E4: Partitioning Exclusively	2/2 on T1 3/3 on T1 ACRT_POS1.0_COIL1.2_YIELD2.2 ACRT_POS1.0_COIL1.3_YIELD3.3	1/2 on T2 2/2 on T2 2/3 on T2[1] ACRT_POS1.2_COIL1.2_YIELD1.2 ACRT_POS1.2_COIL1.2_YIELD2.2 ACRT_POS1.2_COIL1.3_YIELD2.3	2/2 on T3 3/3 on T3 ACRT_POS2.2_COIL1.2_YIELD2.2 ACRT_POS2.2_COIL1.3_YIELD3.3	1/2 on T4 2/3 on T4[1] ACRT_POS4.2_COIL1.2_YIELD1.2 ACRT_POS4.2_COIL1.3_YIELD2.3

[1] These actions were identified as belonging to this cluster by the analysis but do not appear to belong to this error pattern.

Table B9

Stage 4 – Level 5 Clusters

	Trampoline			
Cluster	T1	T2	T3	T4
S1: Standard Solution	1/1 on T1 ACRT_POS1.1_COIL1.1_YIELD1.1	1/1 on T2 ACRT_POS1.3_COIL1.1_YIELD1.1	1/2 on T3 ACRT_POS3.3_COIL1.2_YIELD1.2	1/2 on T4 2/2 on T4 ACRT_POS3.2_COIL1.2_YIELD1.2 ACRT_POS3.2_COIL1.2_YIELD2.2
S2: Alternate Solution	2/2 on T1 ACRT_POS1.1_COIL1.2_YIELD2.2	2/2 on T2 ACRT_POS1.3_COIL1.2_YIELD2.2		1/1 on T4 ACRT_POS3.2_COIL1.1_YIELD1.1
E3: Unitizing Errors	1/2 on T1 ACRT_POS1.1_COIL1.2_YIELD1.2	1/2 on T2 ACRT_POS1.3_COIL1.2_YIELD1.2	1/4 on T3 ACRT_POS3.3_COIL1.4_YIELD1.4	

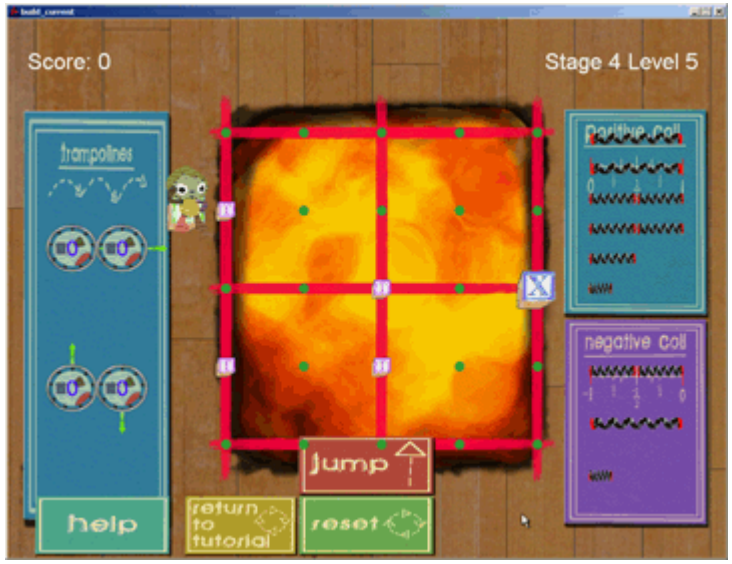


Table B10

Stage 5 – Level 1 Clusters

	Trampoline	
Cluster	T1	OTHER
S1: Standard Solution	1/2 on T1 2/2 on T1 3/2 on T1 ACRT_POS1.0_COIL1.2_YIELD1.2 ACRT_POS1.0_COIL1.2_YIELD2.2 ACRT_POS1.0_COIL1.2_YIELD3.2	
S3: Larger Denominator	1/4 on T1 2/4 on T1 3/4 on T1 4/4 on T1 5/4 on T1 6/4 on T1 ACRT_POS1.0_COIL1.4_YIELD1.4 ACRT_POS1.0_COIL1.4_YIELD2.4 ACRT_POS1.0_COIL1.4_YIELD3.4 ACRT_POS1.0_COIL1.4_YIELD4.4 ACRT_POS1.0_COIL1.4_YIELD5.4 ACRT_POS1.0_COIL1.4_YIELD6.4	
E1: All in Order	2/1 on T1 ACRT_POS1.0_COIL1.1_YIELD2.1	
E3: Unitizing Errors	1/3 on T1 2/3 on T1 3/3 on T1 4/3 on T1 ACRT_POS1.0_COIL1.3_YIELD1.3 ACRT_POS1.0_COIL1.3_YIELD2.3 ACRT_POS1.0_COIL1.3_YIELD3.3 ACRT_POS1.0_COIL1.3_YIELD4.3	

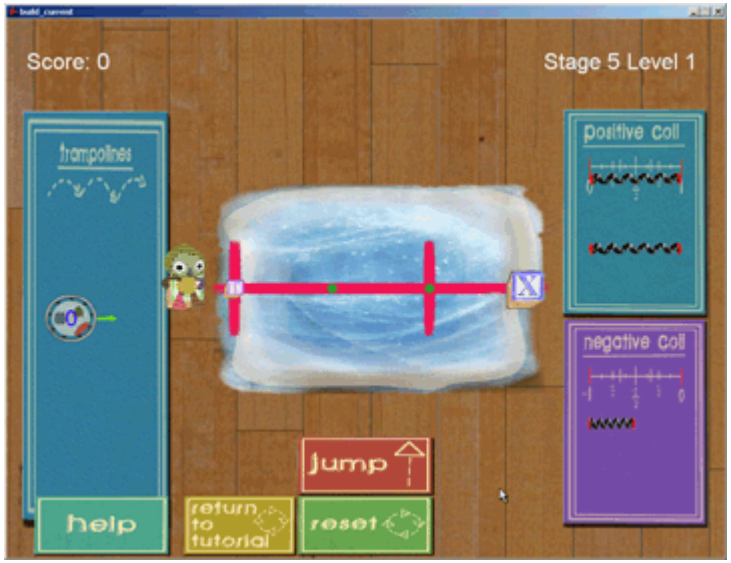
	Trampoline	
Cluster	T1	OTHER
E6: See as Mixed Number	1/1 on T1 ACRT_POS1.0_COIL1.1_YIELD1.1	ADD 1/2 to 1/1 ADD 1/3 to 1/1 ADD 1/4 to 1/1 ADD 1/5 to 1/1 ACWR_COIL1.2_TRAMP1.1 ACWR_COIL1.3_TRAMP1.1 ACWR_COIL1.4_TRAMP1.1 ACWR_COIL1.5_TRAMP1.1
		

Table B11

Stage 5 – Level 2 Clusters

Cluster	Trampoline		
	T1	T2	T3
S1: Standard Solution	1/3 on T1 2/3 on T1 ACRT_POS1.0_COIL1.3_YIELD1.3 ACRT_POS1.0_COIL1.3_YIELD2.3	1/3 on T2 2/3 on T2 ACRT_POS1.2_COIL1.3_YIELD1.3 ACRT_POS1.2_COIL1.3_YIELD2.3	1/3 on T3 ACRT_POS3.2_COIL1.3_YIELD1.3
E4: Partitioning Exclusively	1/2 on T1 2/2 on T1 1/1 on T1 ACRT_POS1.0_COIL1.2_YIELD1.2 ACRT_POS1.0_COIL1.2_YIELD2.2 ACRT_POS1.0_COIL1.1_YIELD1.1	1/2 on T2 2/2 on T2 1/1 on T2 ACRT_POS1.2_COIL1.2_YIELD1.2 ACRT_POS1.2_COIL1.2_YIELD2.2 ACRT_POS1.2_COIL1.1_YIELD1.1	1/2 on T3 ACRT_POS3.2_COIL1.2_YIELD1.2
E5: Partitioning Inclusively	1/4 on T1 2/4 on T1 3/4 on T1 ACRT_POS1.0_COIL1.4_YIELD1.4 ACRT_POS1.0_COIL1.4_YIELD2.4 ACRT_POS1.0_COIL1.4_YIELD3.4	1/4 on T2 2/4 on T2 3/4 on T2 ACRT_POS1.2_COIL1.4_YIELD1.4 ACRT_POS1.2_COIL1.4_YIELD2.4 ACRT_POS1.2_COIL1.4_YIELD3.4	1/4 on T3 ACRT_POS3.2_COIL1.4_YIELD1.4

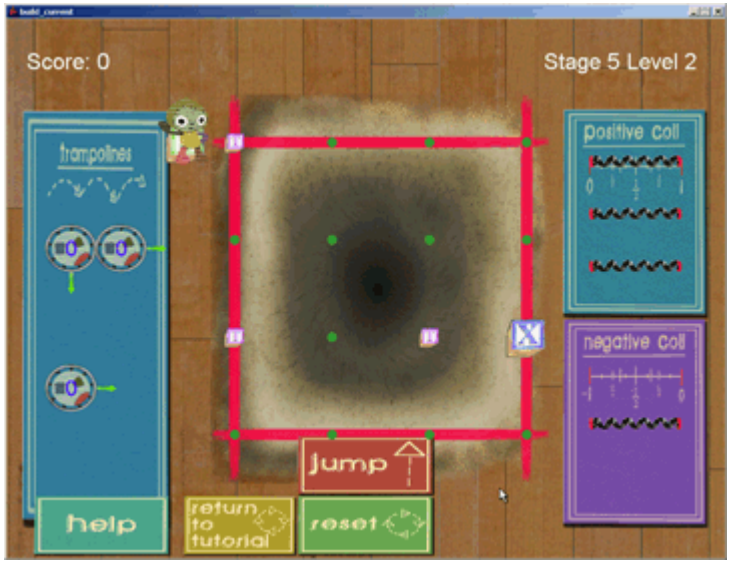
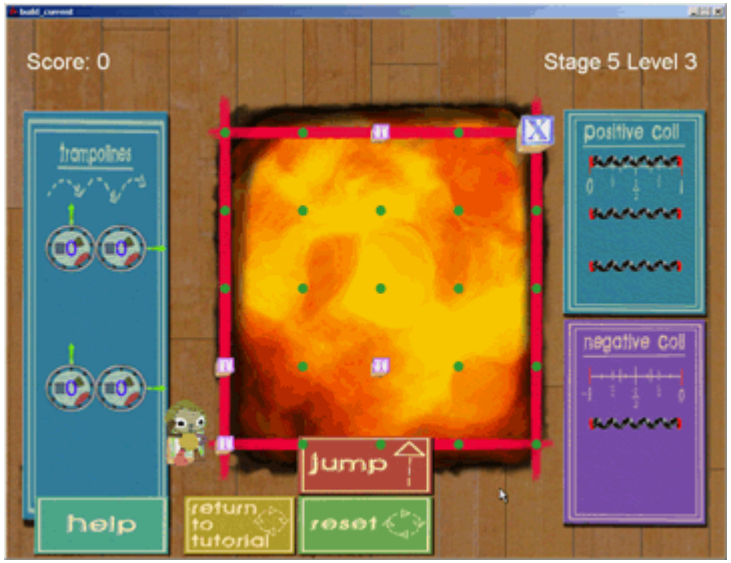

Score: 0
Stage 5 Level 2
trampolines
positive coil
negative coil
Jump
help
return to tutorial
reset

Table B12

Stage 5 – Level 3 Clusters

	Trampoline			
Cluster	T1	T2	T3	T4
S1: Standard Solution	1/4 on T1 ACRT_POS1.4_COIL1.4_YIELD1.4	1/4 on T2 2/4 on T2 ACRT_POS1.3_COIL1.4_YIELD1.4 ACRT_POS1.3_COIL1.4_YIELD2.4	1/4 on T3 2/4 on T3 3/4 on T3 ACRT_POS3.3_COIL1.4_YIELD1.4 ACRT_POS3.3_COIL1.4_YIELD2.4 ACRT_POS3.3_COIL1.4_YIELD3.4	1/4 on T4 2/4 on T4 ACRT_POS3.0_COIL1.4_YIELD1.4 ACRT_POS3.0_COIL1.4_YIELD2.4
S2: Alternate Solution	1/2 on T1[1] ACRT_POS1.4_COIL1.2_YIELD1.2	1/2 on T2 2/2 on T2[1] ACRT_POS1.3_COIL1.2_YIELD1.2 ACRT_POS1.3_COIL1.2_YIELD2.2		1/2 on T4 2/2 on T4[1] ACRT_POS3.0_COIL1.2_YIELD1.2 ACRT_POS3.0_COIL1.2_YIELD2.2
E4: Partitioning Exclusively	1/3 on T1 ACRT_POS1.4_COIL1.3_YIELD1.3	1/3 on T2 2/3 on T2 ACRT_POS1.3_COIL1.3_YIELD1.3 ACRT_POS1.3_COIL1.3_YIELD2.3	1/2 on T3 2/2 on T3 ACRT_POS3.3_COIL1.2_YIELD1.2 ACRT_POS3.3_COIL1.2_YIELD2.2	1/1 on T3 ACRT_POS3.0_COIL1.1_YIELD1.1
E5: Partitioning Inclusively	1/5 on T1 ACRT_POS1.4_COIL1.5_YIELD1.5	1/5 on T2 2/5 on T2 ACRT_POS1.3_COIL1.5_YIELD1.5 ACRT_POS1.3_COIL1.5_YIELD2.5	1/5 on T3 2/5 on T3 3/5 on T3 4/5 on T3[1] ACRT_POS3.3_COIL1.5_YIELD1.5 ACRT_POS3.3_COIL1.5_YIELD2.5 ACRT_POS3.3_COIL1.5_YIELD3.5 ACRT_POS3.3_COIL1.5_YIELD4.5	1/5 on T4 2/5 on T4 ACRT_POS3.0_COIL1.5_YIELD1.5 ACRT_POS3.0_COIL1.5_YIELD2.5



[1]These actions were identified as belonging to this cluster by the analysis but do not appear to belong to this error pattern.

Table B13

Stage 6 – Level 1 Clusters

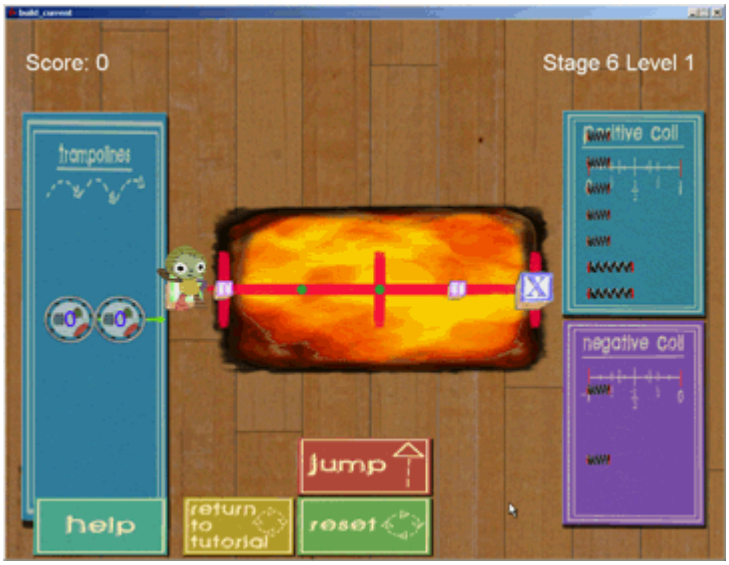
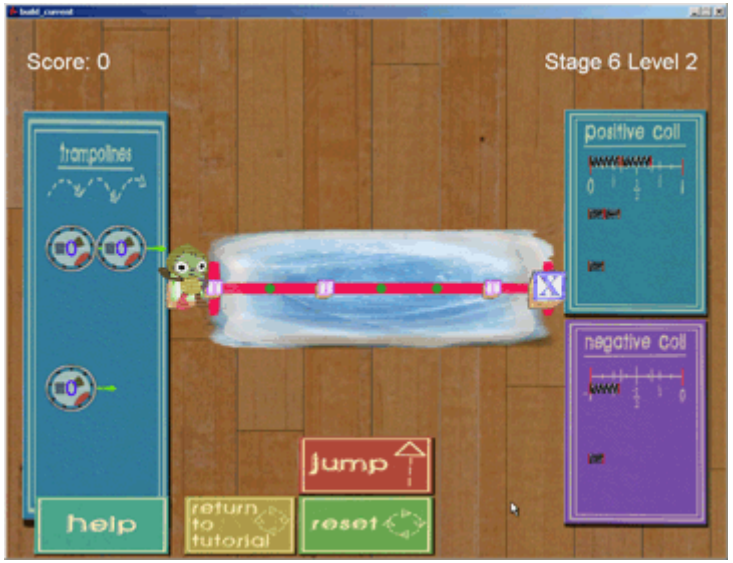
Cluster	Trampoline	
	T1	T2
S1: Standard Solution	4/4 on T1 5/4 on T1 6/4 on T1 ACRT_POS1.0_COIL1.4_YIELD4.4 ACRT_POS1.0_COIL1.4_YIELD5.4 ACRT_POS1.0_COIL1.4_YIELD6.4	2/4 on T2 ACRT_POS4.0_COIL1.4_YIELD2.4
E2: Misusing Resources	1/2 on T1 2/2 on T1 ACRT_POS1.0_COIL1.2_YIELD1.2 ACRT_POS1.0_COIL1.2_YIELD2.2	
E3: Unitizing Errors	1/4 on T1 2/4 on T1 3/4 on T1 ACRT_POS1.0_COIL1.4_YIELD1.4 ACRT_POS1.0_COIL1.4_YIELD2.4 ACRT_POS1.0_COIL1.4_YIELD3.4	1/4 on T2 ACRT_POS4.0_COIL1.4_YIELD1.4
		

Table B14

Stage 6 – Level 2 Clusters

	Trampoline			
Cluster	T1	T2	T3	OTHER
S1: Standard Solution	1/6 on T1 2/6 on T1 ACRT_POS1.0_COIL1.6_YIELD1.6 ACRT_POS1.0_COIL1.6_YIELD2.6	1/6 on T2 2/6 on T2 3/6 on T2 ACRT_POS3.0_COIL1.6_YIELD1.6 ACRT_POS3.0_COIL1.6_YIELD2.6 ACRT_POS3.0_COIL1.6_YIELD3.6	1/6 on T3 ACRT_POS6.0_COIL1.6_YIELD1.6	
S2: Alternate Solution	1/3 on T1 ACRT_POS1.0_COIL1.3_YIELD1.3	4/6 on T2 ACRT_POS3.0_COIL1.6_YIELD4.6		REMOVE T1[1] RMVTRMP_VAL0.1_POS1.0
S4: Shortcut Solution	3/6 on T1 4/6 on T1 5/6 on T1 6/6 on T1 ACRT_POS1.0_COIL1.6_YIELD3.6 ACRT_POS1.0_COIL1.6_YIELD4.6 ACRT_POS1.0_COIL1.6_YIELD5.6 ACRT_POS1.0_COIL1.6_YIELD6.6	1/3 on T2 2/3 on T2 ACRT_POS3.0_COIL1.3_YIELD1.3 ACRT_POS3.0_COIL1.3_YIELD2.3		
E1: All in Order	2/3 on T1 ACRT_POS1.0_COIL1.3_YIELD2.3		2/6 on T3 ACRT_POS6.0_COIL1.6_YIELD2.6	ADD 1/6 to 1/3 ACWR_COIL1.6_TRAMP1.3



[1]These actions were identified as belonging to this cluster by the analysis but do not appear to belong to this error pattern.

Table B15

Stage 6 – Level 3 Clusters

Cluster	Trampoline		
	T1	T2	T3
S1: Standard Solution	1/3 on T1 2/3 on T1 ACRT_POS1.0_COIL1.3_YIELD1.3 ACRT_POS1.0_COIL1.3_YIELD2.3	1/6 on T2 2/6 on T2 3/6 on T2 4/6 on T2 ACRT_POS1.2_COIL1.6_YIELD1.6 ACRT_POS1.2_COIL1.6_YIELD2.6 ACRT_POS1.2_COIL1.6_YIELD3.6 ACRT_POS1.2_COIL1.6_YIELD4.6	1/6 on T3 2/6 on T3 ACRT_POS3.2_COIL1.6_YIELD1.6 ACRT_POS3.2_COIL1.6_YIELD2.6
S2: Alternate Solution	1/6 on T1 2/6 on T1 3/6 on T1 4/6 on T1 ACRT_POS1.0_COIL1.6_YIELD1.6 ACRT_POS1.0_COIL1.6_YIELD2.6 ACRT_POS1.0_COIL1.6_YIELD3.6 ACRT_POS1.0_COIL1.6_YIELD4.6	1/3 on T2 2/3 on T2 ACRT_POS1.2_COIL1.3_YIELD1.3 ACRT_POS1.2_COIL1.3_YIELD2.3	1/3 on T ACRT_POS3.2_COIL1.3_YIELD1.3
S4: Shortcut Solution		5/6 on T2 6/6 on T2 ACRT_POS1.2_COIL1.6_YIELD5.6 ACRT_POS1.2_COIL1.6_YIELD6.6	
E4: Partitioning Exclusively	3/3 on T1 ACRT_POS1.0_COIL1.3_YIELD3.3	3/3 on T2 1/1 on T2 ACRT_POS1.2_COIL1.3_YIELD3.3 ACRT_POS1.2_COIL1.1_YIELD1.1	3/6 on T3 ACRT_POS3.2_COIL1.6_YIELD3.6
E5: Partitioning Inclusively	1/4 on T1 2/4 on T1 ACRT_POS1.0_COIL1.4_YIELD1.4 ACRT_POS1.0_COIL1.4_YIELD2.4	1/4 on T2 2/4 on T2 ACRT_POS1.2_COIL1.4_YIELD1.4 ACRT_POS1.2_COIL1.4_YIELD2.4	1/4 on T3 ACRT_POS3.2_COIL1.4_YIELD1.4

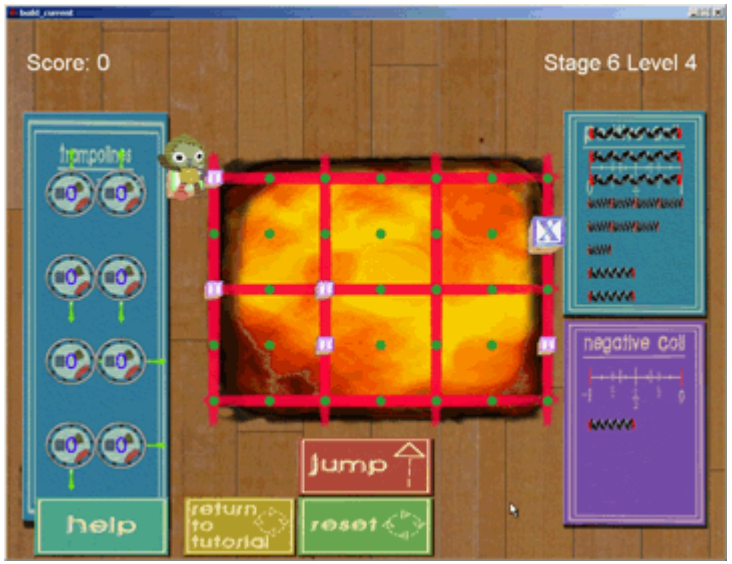

Score: 0
Stage 6 Level 3
trampolines
Positive Coil
negative Coil
Jump
help
return to tutorial
reset

Table B16

Stage 6 – Level 4 Clusters

	Trampoline				
Cluster	T1	T2	T3	T4	T5
S1: Standard Solution	1/1 on T1 ACRT_POS1.0_ COIL1.1_YIELD1.1	1/1 on T2 ACRT_POS1.2_COIL1.1_YIELD1.1	1/2 on T3 ACRT_POS3.2_ COIL1.2_YIELD1.2	1/4 on T4 2/4 on T4 3/4 on T4 4/4 on T4 5/4 on T4 6/4 on T4 7/4 on T4 8/4 on T4 ACRT_POS3.3_COIL1.4_YIELD1.4 ACRT_POS3.3_COIL1.4_YIELD2.4 ACRT_POS3.3_COIL1.4_YIELD3.4 ACRT_POS3.3_COIL1.4_YIELD4.4 ACRT_POS3.3_COIL1.4_YIELD5.4 ACRT_POS3.3_COIL1.4_YIELD6.4 ACRT_POS3.3_COIL1.4_YIELD7.4 ACRT_POS3.3_COIL1.4_YIELD8.4	1/1 on T5 ACRT_POS7.3_COIL1.1_YIELD1.1
S2: Alternate Solution	3/4 on T1 4/4 on T1 ACRT_POS1.0_ COIL1.4_YIELD3.4 ACRT_POS1.0_ COIL1.4_YIELD4.4	4/4 on T2 ACRT_POS1.2_COIL1.4_YIELD4.64		1/1 on T4 2/1 on T4 1/2 on T4 2/2 on T4 3/2 on T4 4/2 on T4 ACRT_POS3.3_COIL1.1_YIELD1.1 ACRT_POS3.3_COIL1.1_YIELD2.1 ACRT_POS3.3_COIL1.2_YIELD1.2 ACRT_POS3.3_COIL1.2_YIELD2.2 ACRT_POS3.3_COIL1.2_YIELD3.2 ACRT_POS3.3_COIL1.2_YIELD4.2	1/4 on T5 2/4 on T5 3/4 on T5 4/4 on T5 1/2 on T5 2/2 on T5 ACRT_POS7.3_COIL1.4_YIELD1.4 ACRT_POS7.3_COIL1.4_YIELD2.4 ACRT_POS7.3_COIL1.4_YIELD3.4 ACRT_POS7.3_COIL1.4_YIELD4.4 ACRT_POS7.3_COIL1.2_YIELD1.2 ACRT_POS7.3_COIL1.2_YIELD2.2

	Trampoline				
Cluster	T1	T2	T3	T4	T5
E2: Misusing Resources			1/4 on T3 2/4 on T3 ACRT_POS3.2_ COIL1.4_YIELD1.4 ACRT_POS3.2_ COIL1.4_YIELD2.4		
E3: Unitizing Errors	1/4 on T1 2/4 on T1 ACRT_POS1.0_ COIL1.4_YIELD1.4 ACRT_POS1.0_ COIL1.4_YIELD2.4	1/4 on T2 2/4 on T2 3/4 on T2[1] ACRT_POS1.2_COIL1.4_YIELD1.4 ACRT_POS1.2_COIL1.4_YIELD2.4 ACRT_POS1.2_COIL1.4_YIELD3.4			
E5: Partitioning Inclusively		1/6 on T2 2/6 on T2 ACRT_POS1.2_COIL1.6_YIELD1.6 ACRT_POS1.2_COIL1.6_YIELD2.6		1/6 on T4 2/6 on T4 3/6 on T4 4/6 on T4 ACRT_POS3.3_COIL1.6_YIELD1.6 ACRT_POS3.3_COIL1.6_YIELD2.6 ACRT_POS3.3_COIL1.6_YIELD3.6 ACRT_POS3.3_COIL1.6_YIELD4.6	

	Trampoline				
Cluster	T1	T2	T3	T4	T5
					

[1]These actions were identified as belonging to this cluster by the analysis but do not appear to belong to this error pattern.

Appendix C:

SPSS Syntax

The following code was used to compute the mnemonic in SPSS. The mnemonic is a variable that combines information from a number of different variables in the log data so that all relevant information about a student's actions can be displayed in one column.

```
**************************PART 1************************.
* compute mnemonic values for data code 2010.
* ACRT_POSxy_COILnd_YIELDnd.
string x (a20).
string y (a20).
string n1 (a20).
string n2 (a20).
string d1 (a20).
string d2 (a20).
string mnemonic (a100).
string strbase1 (a100).
string strbase2 (a100).
string strbase3 (a100).
string separator (a1).
string ss1 (a20).
string ss2 (a20).
string ss3 (a20).
string s1 (a20).
string s2 (a20).
string s3 (a20).
string s4 (a20).
exe.

* char position.
compute strbase1 = 'ACRT_POS'.
compute strbase2 = '_COIL'.
compute strbase3 = '_YIELD'.
compute separator = '.'.

* set up strings to delimit event data.
compute s1 = 'position'.
compute s2 = 'added'.
compute s3 = 'yield'.
compute s4 = 'to'.
compute len1 = char.length(s1).
compute len2 = char.length(s2).
compute len3 = char.length(s3).
```

```
exe.

do if event_code = 2010.
  * at position 1 0 added 1/2 to yield 1/2.

  * find position of 'position', 'added' and 'yield'. Will strip these out
  * and pull out digits from the substrings.
  compute p1 = char.index(event_data, rtrim(s1)) + len1 + 1.
  compute p2 = char.index(event_data, rtrim(s2)).
  compute p3 = char.index(event_data, rtrim(s3)).
  compute p4 = char.index(event_data, rtrim(s4)).

  * pull out substrings.
  compute ss1 = char.substr(event_data, p1, p2 - p1).
  compute ss2 = char.substr(event_data, p2 + len2 + 1, p4 - (p2 + len2 + 1)).
  compute ss3 = char.substr(event_data, p3 + len3 + 1).

  * find delimiters within substring for each component.
  compute ssp1 = char.index(ss1, ' '). /* space separate x and y.
  compute ssp2 = char.index(ss2, '/'). /* / separates numerator and denom.
  compute ssp3 = char.index(ss3, '/'). /* / separates numerator and denom.

  * pull out component digits.
  compute x =   char.substr(ss1, 1, ssp1 - 1).
  compute y =   char.substr(ss1, ssp1 + 1).

  compute n1  = char.substr(ss2, 1, ssp2 - 1).
  compute d1  = char.substr(ss2, ssp2 + 1).

  compute n2  = char.substr(ss3, 1, ssp3 - 1).
  compute d2  = char.substr(ss3, ssp3 + 1).

  * check for numeric values.
  compute temp_x  = number(x,F2.0).
  compute temp_y  = number(y,F2.0).
  compute temp_n1 = number(n1,F2.0).
  compute temp_n2 = number(n2,F2.0).
  compute temp_d1 = number(d1,F2.0).
  compute temp_d2 = number(d2,F2.0).

  if missing(temp_x)   x = '?'.
  if missing(temp_y)   y = '?'.
  if missing(temp_n1) n1 = '?'.
  if missing(temp_n2) n2 = '?'.
  if missing(temp_d1) d1 = '?'.
  if missing(temp_d2) d2 = '?'.
```

```
   compute mnemonic = concat(rtrim(strbase1), rtrim(x),  separator, rtrim(y),
                       rtrim(strbase2), rtrim(n1), separator, rtrim(d1),
                       rtrim(strbase3), rtrim(n2), separator, rtrim(d2)).
end if.
exe.

* clean up.
delete var
x, y, n1, n2, d1, d2, strbase1, strbase2, strbase3, separator
ss1, ss2, ss3, s1, s2, s3, s4, len1, len2,len3, p1, p2, p3, p4
ssp1, ssp2, ssp3, temp_x, temp_y, temp_n1, temp_n2, temp_d1, temp_d2

*****************************PART 2************************.
* compute mnemonic values for data code 2011.
* ACWR_COILnd_TRAMPnd.
string n1 (a20).
string d1 (a20).
string n2 (a20).
string d2 (a20).
string mnemonic (a100).
string strbase1 (a100).
string strbase2 (a100).
string separator (a1).
string ss1 (a20).
string ss2 (a20).
string s1 (a20).
string s2 (a20).
exe.

* char position.
compute strbase1 = 'ACWR_COIL'.
compute strbase2 = '_TRAMP'.
compute separator = '.'.

* set up strings to delimit event data.
compute s1 = 'add'.
compute s2 = 'to'.

compute len1 = char.length(s1).
compute len2 = char.length(s2).
exe.

do if event_code = 2011.
* did not work because user tried to add 1/2 to 1/4.
```

```
  * find position of 'value' and 'point'. Will strip these out
  * and pull out digits from the substrings.
  * p1 = start of fraction '1/2'
  * p2 = start of fraction '1/4'.

  compute p1 = char.index(event_data, rtrim(s1)) + len1 + 1.
  compute p2 = char.rindex(event_data, rtrim(s2)) + len2 + 1.

  * pull out substrings.
  compute ss1 = char.substr(event_data, p1, p2 - p1 - len2 - 1).
  compute ss2 = char.substr(event_data, p2).

  * find delimiters within substring for each component.
  compute ssp1 = char.index(ss1, '/'). /* / separates numerator and denom.

  * pull out component digits.
  compute n1  = char.substr(ss1, 1, ssp1 - 1).
  compute d1  = char.substr(ss1, ssp1 + 1).
  compute n2  = char.substr(ss2, 1, ssp1 - 1).
  compute d2  = char.substr(ss2, ssp1 + 1).

  * check for numeric values.
  compute temp_n1  = number(n1,F2.0).
  compute temp_d1  = number(d1,F2.0).
  compute temp_n2 = number(n2,F2.0).
  compute temp_d2 = number(d2,F2.0).

  if missing(temp_n1)   n1 = '?'.
  if missing(temp_d1)   d1 = '?'.
  if missing(temp_n2)   n2 = '?'.
  if missing(temp_d2)   d2 = '?'.

  compute mnemonic = concat(rtrim(strbase1), rtrim(n1),  separator, rtrim(d1),
                    rtrim(strbase2), rtrim(n2), separator, rtrim(d2)).
end if.
exe.

* clean up.
delete var
n1, d1, n2, d2, strbase1, strbase2, separator
ss1, ss2, s1, s2, len1, len2, p1, p2, ssp1
temp_n1, temp_d1, temp_n2, temp_d2

**************************PART 3************************.
* compute mnemonic values for data code 2120.
* PLCTRMP_VALnd_POSxy.
```

```
string n (a20).
string d (a20).
string x (a20).
string y (a20).
string mnemonic (a100).
string strbase1 (a100).
string strbase2 (a100).
string separator (a1).
string ss1 (a20).
string ss2 (a20).
string s1 (a20).
string s2 (a20).
string s3 (a20).
exe.

* char position.
compute strbase1 = 'PLCTRMP_VAL'.
compute strbase2 = '_POS'.
compute separator = '.'.

* set up strings to delimit event data.
compute s1 = 'value'.
compute s2 = 'point'.
compute s3 = 'to'.

compute len1 = char.length(s1).
compute len2 = char.length(s2).
compute len3 = char.length(s3).
exe.

do if event_code = 2120.
  * placed a trampoline of value 2/2 to point 1 0.

  * find position of 'value' and 'point'. Will strip these out
  * and pull out digits from the substrings.
  * p1 = start of fraction '2/2'
  * p2 = start of 'point'
  * p3 = start of 'to'.

  compute p1 = char.index(event_data, rtrim(s1)) + len1 + 1.
  compute p2 = char.index(event_data, rtrim(s2)).
  compute p3 = char.index(event_data, rtrim(s3)).

  * pull out substrings.
  compute ss1 = char.substr(event_data, p1, p3 - p1).
  compute ss2 = char.substr(event_data, p2 + len2 + 1).
```

```
   * find delimiters within substring for each component.
   compute ssp1 = char.index(ss1, '/'). /* / separates numerator and denom.
   compute ssp2 = char.index(ss2, ' '). /* space separate x and y.

   * pull out component digits.
   compute n  = char.substr(ss1, 1, ssp1 - 1).
   compute d  = char.substr(ss1, ssp1 + 1).
   compute x  = char.substr(ss2, 1, ssp2 - 1).
   compute y  = char.substr(ss2, ssp2 + 1).

   * check for numeric values.
   compute temp_n  = number(n,F2.0).
   compute temp_d  = number(d,F2.0).
   compute temp_x = number(x,F2.0).
   compute temp_y = number(y,F2.0).

   if missing(temp_x)   x = '?'.
   if missing(temp_y)   y = '?'.
   if missing(temp_n)   n = '?'.
   if missing(temp_d)   d = '?'.

   compute mnemonic = concat(rtrim(strbase1), rtrim(n),  separator, rtrim(d),
                        rtrim(strbase2), rtrim(x), separator, rtrim(y)).
end if.
exe.

* clean up.
delete var
x, y, n, d, strbase1, strbase2, separator
ss1, ss2, s1, s2, s3, len1, len2,len3, p1, p2, p3
ssp1, ssp2, temp_x, temp_y, temp_n, temp_d

**************************PART 4************************.
* compute mnemonic values for data code 2020.
* RMVTRMP_VALnd_POSxy.
string n (a20).
string d (a20).
string x (a20).
string y (a20).
string mnemonic (a100).
string strbase1 (a100).
string strbase2 (a100).
string separator (a1).
string ss1 (a20).
string ss2 (a20).
```

```
string s1 (a20).
string s2 (a20).
string s3 (a20).
exe.

* char position.
compute strbase1 = 'RMVTRMP_VAL'.
compute strbase2 = '_POS'.
compute separator = '.'.

* set up strings to delimit event data.
compute s1 = 'value'.
compute s2 = 'point'.
compute s3 = 'from'.

compute len1 = char.length(s1).
compute len2 = char.length(s2).
compute len3 = char.length(s3).
exe.

do if event_code = 2020.
  * removed a trampoline of value 0/1 from point 1 0.

  * find position of 'value' and 'point'. Will strip these out
  * and pull out digits from the substrings.
  * p1 = start of fraction '0/1'
  * p2 = start of 'point'
  * p3 = start of 'from'.

  compute p1 = char.index(event_data, rtrim(s1)) + len1 + 1.
  compute p2 = char.index(event_data, rtrim(s2)).
  compute p3 = char.index(event_data, rtrim(s3)).

  * pull out substrings.
  compute ss1 = char.substr(event_data, p1, p3 - p1).
  compute ss2 = char.substr(event_data, p2 + len2 + 1).

  * find delimiters within substring for each component.
  compute ssp1 = char.index(ss1, '/'). /* / separates numerator and denom.
  compute ssp2 = char.index(ss2, ' '). /* space separate x and y.

  * pull out component digits.
  compute n  = char.substr(ss1, 1, ssp1 - 1).
  compute d  = char.substr(ss1, ssp1 + 1).
  compute x  = char.substr(ss2, 1, ssp2 - 1).
  compute y  = char.substr(ss2, ssp2 + 1).
```

```
   * check for numeric values.
   compute temp_n  = number(n,F2.0).
   compute temp_d  = number(d,F2.0).
   compute temp_x = number(x,F2.0).
   compute temp_y = number(y,F2.0).

   if missing(temp_x)   x = '?'.
   if missing(temp_y)   y = '?'.
   if missing(temp_n)   n = '?'.
   if missing(temp_d)   d = '?'.

   compute mnemonic = concat(rtrim(strbase1), rtrim(n),  separator, rtrim(d),
                        rtrim(strbase2), rtrim(x), separator, rtrim(y)).
end if.
exe.

* clean up.
delete var
x, y, n, d, strbase1, strbase2, separator
ss1, ss2, s1, s2, s3, len1, len2,len3, p1, p2, p3
ssp1, ssp2, temp_x, temp_y, temp_n, temp_d

**************************PART 5************************.
* add remaining variables of interest not captured in event_data.
if event_code = 2062 mnemonic = "RET_TUT".
if event_code = 2030 mnemonic = "TOG_TRMP".
exe.
```

Appendix D:
R Code

The following code was used to run fuzzy cluster analysis in R.

```
# run the following to import level data from a text file into R
twentyfour <- read.table("file_address", header = TRUE)
express <- twentyfour[, c( "ACRT_POS3.2_COIL1.4_YIELD1.4", etc.)]

# run the following to call the necessary libraries
# cluster loads the fanny function
# vegan loads the ord, ordiplot, stars, and ordihull functions
library(cluster)
library(vegan)

# run the following to transpose the matrix so that the actions are analyzed
# otherwise the students will be clustered instead of the actions
expresst <- t(express)

# run the following to create the distance matrix
dexpt <- dist(expresst,"manhattan")

# run the following to determine how the clusters are composed
# run numerous times, with the value after dexpt ranging from 2 to (n/2)-1
f <- fanny(dexpt,2,TRUE,2,"manhattan",TRUE,NULL,FALSE,TRUE,TRUE,
550,1e-15,0)
summary(f)

# once the correct number of clusters has been determined
# run the following to plot the clusters
ord <- cmdscale(dexpt)
ordiplot(ord, dis = "si")
ordihull(ord, f$clustering, col = "blue")
stars(f$membership, locatio = ord, draw.segm = TRUE, add = TRUE, scale = FALSE,
len = 0.1, xpd = TRUE)
```

Appendix E:
Percentage of Attempts in Each Cluster

Four solution strategies were identified: a standard solution (Solution 1), an alternate solution (Solution 2), a solution using a larger denominator (Solution 3), and a solution using a shortcut (Solution 4). Six error patterns were identified: using all resources in the order in which they were provided (Error 1), misusing resources (Error 2), unitizing errors (Error 3), partitioning exclusively (Error 4), partitioning inclusively (Error 5), and seeing the solution as a mixed number (Error 6). Table E1 displays the detected strategies and errors for each level. Asterisks indicate levels in which a given cluster was possible but was not identified by cluster analysis. Solution 3 was technically possible in all levels since students could always scroll to a larger denominator if they wanted, though it was rarely used. Stage 1 – Level 1 and Stage 2 – Level 2 were not analyzed, as cluster analysis did not identify any clusters besides the standard solution.

Table E1

Percentage of Attempts in Each Cluster

	Solutions				Error patterns					
Stage - Level	S1	S2	S3	S4	E1	E2	E3	E4	E5	E6
1-1	87%	--	--	--	--	--	--	--	--	--
2-1	64%	*	--	*	--	4%	3%	--	--	--
2-2	82%	--	--	--	--	--	--	--	--	--
3-1	51%	--	--	--	37%	--	--	--	--	--
3-2	6%	--	--	*	--	--		68%	*	--
3-3	45%	--	--	--	17%	9%	*	*	*	--
4-1	57%	*	--	--	*	*	29%	--	--	--
4-2	33%	--	--	5%	--	14%	*	*	20%	--
4-3	40%	--	8%	*	32%	2%	--	*	*	--
4-4	15%	--	--	*	7%	--	--	46%	*	--
4-5	20%	19%	--	--	--	--	50%	--	--	--
5-1	19%	--	4%	--	11%	--	9%	--	--	22%
5-2	18%	--	--	*	--	--	--	39%	13%	--
5-3	12%	23%	--	--	--	--	--	24%	6%	--
6-1	13%	*	--	*	*	18%	30%	--	--	*
6-2	23%	31%	--	18%	11%	--	--	*	*	--
6-3	7%	42%	--	2%	*	--	--	18%	5%	--
6-4	3%	15%	--	--	--	19%	27%	--	3%	--