Trying to Reduce Gaming Behavior by Students in Intelligent Tutoring Systems

by

Elijah Forbes-Summers

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

May 2010

APPROVED:

Professor Neil T. Heffernan
Thesis Advisor

Professor Elke A. Rundensteiner
Thesis Reader

Professor Michael A. Gennert
Head of Department

Abstract

Student gaming behavior in intelligent tutoring systems (ITS) has been correlated with lower learning rates. The goal of this work is to identify such behavior, produce interventions to discourage this behavior, and by doing so hopefully improve the learning rate of students who would normally display gaming behavior. Detectors have been built to identify gaming behavior. Interventions have been designed to discourage the behavior and their evaluation is discussed.

Table of Contents

Table of Figures

1. Intelligent Tutoring Systems

An Intelligent Tutoring System (ITS) is a computer-based system designed to simulate in some aspect the activities of a human tutor interacting with students. This usually includes some way of providing customized instruction or feedback to students while they are working within the tutoring system. Intelligent tutoring systems have been built for a variety of domains from reading to physics.

1.1 Assistment System

The ASSISTments system is an ITS and the focus of this work. Students (mostly in middle school) sign into the ASSISTments website and complete math problems in problem sets assigned by their teacher. The ASSISTments tutor assesses the students' performance and provides functions to assist the students with learning the content of problems.

1.2 Assistance Provided by ITSs

A key feature of Intelligent Tutoring Systems is their ability to assist the student in the learning process. Typical forms that this assistance comes in (the main types provided by ASSISTments) are immediate feedback, hints, and scaffolding. Immediate feedback is simply providing some kind of immediate evaluation of the student's activities with the turor. This is usually just an indication of correctness but isn't always. Hints provide a small tip on how to solve the problem the student is working on. Hints could include reminders of properties of the domain or bits of procedural knowledge. In the ASSISTments system students have to request hints. There is a sequence of hints that are available for each problem, each hint gives a little more information about how to solve the current problem, until the last hint (called the "bottom-out-hint") which gives the student the answer to the problem. Because ASSISTments was built as a web-based system and the student may not have someone around to help them, there has to be some mechanic so that the student doesn't become permanently stuck. The final type of assistance is scaffolding. If a student has trouble with a problem many times scaffold problems are available. These problems break down the original problem into conceptually smaller chunks. After completing all the scaffold problems the student has been lead through how to complete all the requisite parts of the original problem and can retry the problem. We will focus on the first two types of assistance I've covered: immediate feedback and hints.

2. Gaming

The intended purpose of this assistance is, of course, to facilitate learning. However, students can find ways of abusing assistance features in order to rapidly progress through content without having to learn the content. For example, the system of progressively revealing hints that I described above which culminates in the answer to the problem is a necessary feature to prevent students from being stuck. Once a student knows that the final hint will provide the answer they can rapidly ask for all the hints. The student will be provided the answer and can quickly move onto the next problem without having to try to solve the current one. Providing immediate feedback can be abused as well. Students can quickly submit different answers until the tutor accepts one. This is easy to imagine happening with multiple choice questions, but students will do this even on open response questions. In general any type of activity that allows students to progress through tutoring content without attempting to solve the problem or learn the content I will refer to as "gaming". This type of behavior is not common however. In [5] students were detected as gaming 14% of the time, and other work in more structured class environments found even lower rates [2].

2.1 Effects of Gaming

Several studies have shown that gaming behavior is negatively correlated with learning [2, 8]. This effect is present at both the aggregate and individual problem level and the difference in learning is dramatic (it is suggested that students who don't game learn 12 times faster than their gaming counterparts) [5].

3. Detecting Gaming Behavior

There has been much previous work attempting to detect gaming behavior [1, 2, 8]. Walonoski, et al. used a variety of different learning techniques to find patterns of actions indicative of gaming using student interaction data that was tagged with human-generated gaming information [8]. One result of this work was a set of machine learned rules for identifying gaming behavior.

3.1 Offline Detection

We designed heuristics based on some of the machine learned rules to build a gaming detector that could tag student log data with gaming information. Criteria for the production of this detector included easily interpretable rules and relatively fast processing so that large amounts of data could be tagged. The approach of using heuristic rules identified in previous work was ideal for meeting these goals.

The detector was tweaked based on its performance on ASSISTment log data. One particularly noteworthy change that was made to the logic of the detector was to implement the notion of a "worked example". This built on work done by Shih, et al. [7] who found that some students who were traditionally being identified as gamers because they asked for all the hints were actually not gaming. When the students came upon a problem they didn't know how to solve they would ask for all the hints (including the answer) and then treat the problem plus expanded hints as a worked example. The students would stop and read through the problem and answer and work out how to do the problem, thus learning from the experience. We incorporated a check for this type of delay after reaching a bottom-out-hint and if we see it the student is not penalized for the rapid hint requests.

Data generated by this detector was used for analysis in [5]. This detector was ideal for producing data for analysis from logs. However, in order to prevent gaming we had to be able to act on the data in real time while students were using the tutor.

3.2 Online Detection

In order to make gaming information available to the tutor a live version of the detector was built. The offline detector was ported to JavaScript and incorporated into the ASSISTments system. The online detector runs on the client-side while the student is working in the ASSISTments system. A gaming estimate is calculated on the fly as the student works. A simple student model on the server-side tracks persistent gaming information between user sessions.

Updated gaming state from the client-side detector is sent to the server by piggybacking on existing AJAX requests carrying information about student actions. This way very little overhead or complexity is added to the functioning of the system. Besides making gaming data available in real-time one of the major benefits of this implementation of the gaming detector is that all the gaming detector is recorded in the student log files along with students actions with the tutor. Because gaming data is calculated for all of the students actions there is no need for other researchers to export data and run the offline detector on it in order to obtain gaming data.

4. Responding to Gaming Behavior

Once we can identify gaming behavior and can detect it in real-time it makes sense that we should try to respond to it in a way that will lessen its occurrence.

4.1 Typical Methods

In [9] Walonoski, et al. used what they termed passive and active interventions. The passive intervention simply made the student aware that the tutor was paying attention to what they were doing. The active interventions directly addressed the student with a message telling them to ask for help or other relevant prompts.

In general we have a few ways of responding when the student games. Letting the student know that the tutor is “watching” them could be enough to discourage gaming behavior, as mentioned above. Another advantage to this passive approach is that this method of communicating the watchfulness of the tutor to the student can also provide queues to a teacher in the room about the student’s actions. This side-effect should not be underestimated. The teacher is one of the greatest resources available to the tutor in this case. It is doubtful that any isolated intervention within the system can be as effective as the physical presence of the teacher. The trick is to provide the proper information to the teacher so that they can identify gaming students and respond appropriately.

We can also take the active approach suggested by Walonoski. These approaches interrupt the student and grab their attention. We can differentiate between active *acknowledging* interventions and *hindering* interventions. The former notifies the student they are being watched, suggest different strategies, etc. The later attempts to prevent further gaming behavior by temporarily disabling or changing the mechanics of the features which allowed gaming in the first place. For example, a hindering intervention might be to disable the student’s ability to request hints for a brief period if we detect the student is abusing the hint feature in order to receive an answer without effort.

In [3] Baker, et al. used a mixture of these types of interventions. They included an animated agent that grew visible displeased when the student was detected as gaming. In addition they forced students who gamed to complete supplemental exercises on the content they gamed on. Their results showed a learning gain on the later intervention, but because those students were forced to complete extra work this is not terribly surprising. Ideally we would like to be able to show a learning gain by *preventing* the normal learning loss associated with gaming.

4.2 Effort Bar

The passive intervention that I have built is the “Effort Bar”. The Effort Bar is a visual representation of the student’s action history. It is designed to clearly show not only the specific interactions the student has with the tutor but also information on how “good” those interactions are. Much of this information is contained within the patterns of points displayed on the bar.

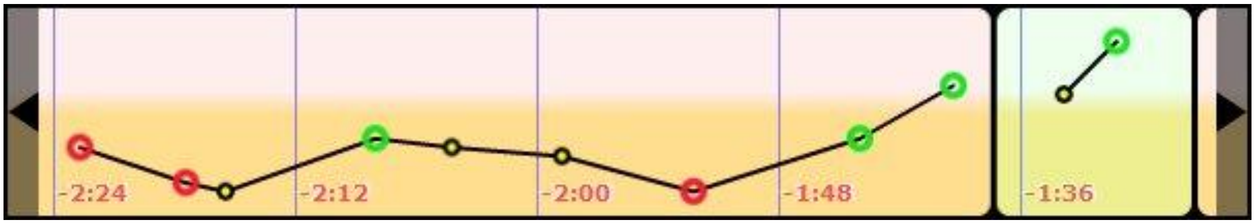


Figure 1: Prototype Effort Bar showing student actions over the course of two problems.

The effort bar lets the student know (in a very persistent and visually obvious way) that they are being watched, even if it is only the computer that is watching. Even more important, however, is that the effort bar allows the teacher to watch the student without constantly monitoring their actions. The teacher can scroll through previous actions, notice patterns of activity (when hints are asked for, how long the student waits between attempted answers, etc.), and see the answers provided by students by inspecting the points on the graph.

Even from a distance the difference between student progressing well (a spaced string of correct green points) and a student abusing the tutor or doing poorly (closely spaced hints or red incorrect points) is distinct. This provides additional queues for the teacher that a student may need help or need to be reminded to stay on task.

4.2.1 Implementation

The effort bar is a flash object written in the open source language haXe. It exposes JavaScript functions that receive updates from the tutor about the student’s actions. It puts no load on the server except for the bandwidth required to initially send the SWF flash file to the client. Several changes were made during the development of the Effort Bar. Most obvious were changes made to its appearance over time in order to maintain visibility without using up the valuable screen real-estate needed by the tutoring content.

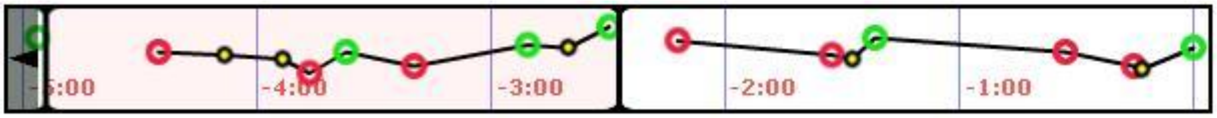


Figure 2: Updated Effort Bar to minimize visual footprint.

4.2.2 Interpreting the Effort Bar

When the Effort Bar is activated in a class it includes a set of instructions which describe the basics of what it displays and how to use it. More importantly, the instructions describe how to interpret the patterns of student activity on the bar.

🟢	Correct Answer
🔴	Wrong Answer
🟡	Hint

Figure 3: Key to Effort Bar points.

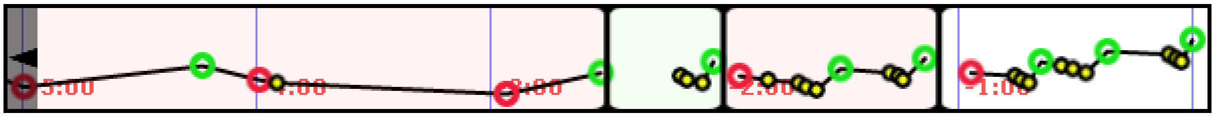


Figure 4: Student "rapid-hinting", getting to the bottom-out-hint to move on.

In Figure 4 the tight clusters of yellow hint points followed by a correct response indicate a student rapidly asking for hints in order to reach the bottom-out-hint and move on without trying to solve the problem.

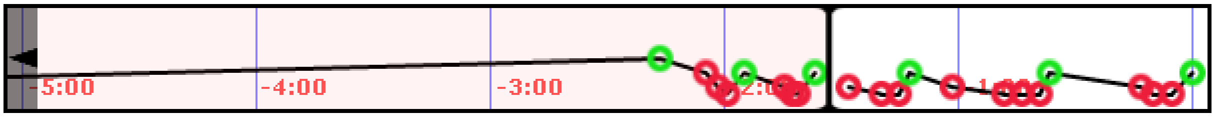


Figure 5: Student "guessing-and-checking", submitting multiple answers until one is correct.

In Figure 5 the clusters of red points indicate multiple incorrect answer submissions. It looks like the student is repeatedly applying this tactic of submitting answers until the tutor accepts one and the student can move on.

4.3 Operant Intervention

I also built an intervention that combines active acknowledging with operant conditioning. We separate this intervention into two different types: positive reinforcement (carrot) and positive punishment (stick). In both cases a modal message dialog is displayed to the student requiring their attention and interaction before they may continue. In the stick type we display the message dialog when students have been tagged as exhibiting gaming behavior by the detector. The message in this case is a stern warning to discontinue their actions. In the carrot type the dialog comes when the student transitions from being tagged as gaming to a non-gaming state. The message in this case has a positive tone and encourages their non-gaming behavior.

Because this intervention is only triggered when the student crosses into or out of the gaming state there are many students who will rarely or never witness this intervention. Also it is possible that students will perceive the messages differently than intended (e.g. carrot message might be seen as patronizing).

5. Study Design

We would like to judge the effectiveness of the implemented interventions outlined above. There are 5 experimental groups corresponding to the individual interventions and their combinations.

		Effort Bar	
		On	Off
Message	Carrot	Group 4	Group 2
	None	Group 1	Control
	Stick	Group 5	Group 3

Figure 6: Study layout.

Each group will be composed of multiple classes. Crossover will be done within group to control for the effect of changing tutoring content as time progresses. Unfortunately using crossover for direct comparison of different groups (i.e. determining which intervention is the absolute best) would require a prohibitive number of additional groups. The importance of this comparison is purely academic however, as the groups selected will provide us with the information about which interventions worked in general (significant effect) and which didn't.

5.1 Measures

Our primary interest is whether we can alter the behavior of the student such that they no longer engage in gaming. Intuitively this should result in increased learning (at least a reduction in the previously observed negative effect of gaming on learning, due to less overall gaming). We will measure learning, but do not expect a large effect (likely not significant) due to the tangential nature of the intervention (it does not directly affect tutoring and does not force any change on the student's part, just hopefully a push in the right direction). However, even without treating the decline of gaming behavior as a proxy for increased learning, we would consider the immediate result a worthy accomplishment. Demonstrating that simple interventions can modify student behavior would provide footing for more complicated behavior change (e.g., teaching students to reflect after receiving a hint; or to better judge their own skill level).

5.2 Measuring Learning

For the purpose of measuring the any learning gain between two groups I use a technique that is common within the Intelligent Tutoring community: Knowledge Tracing [4]. Knowledge Tracing treats each problem the student completes as an opportunity to learn a skill. It represents each student's knowledge of a skill as a latent, i.e., unobservable, variable which influences the student's performance on a problem related to that skill. This is implemented within a Dynamic Bayesian Network.

5.2.1 Knowledge Tracing as a Dynamic Bayesian Network

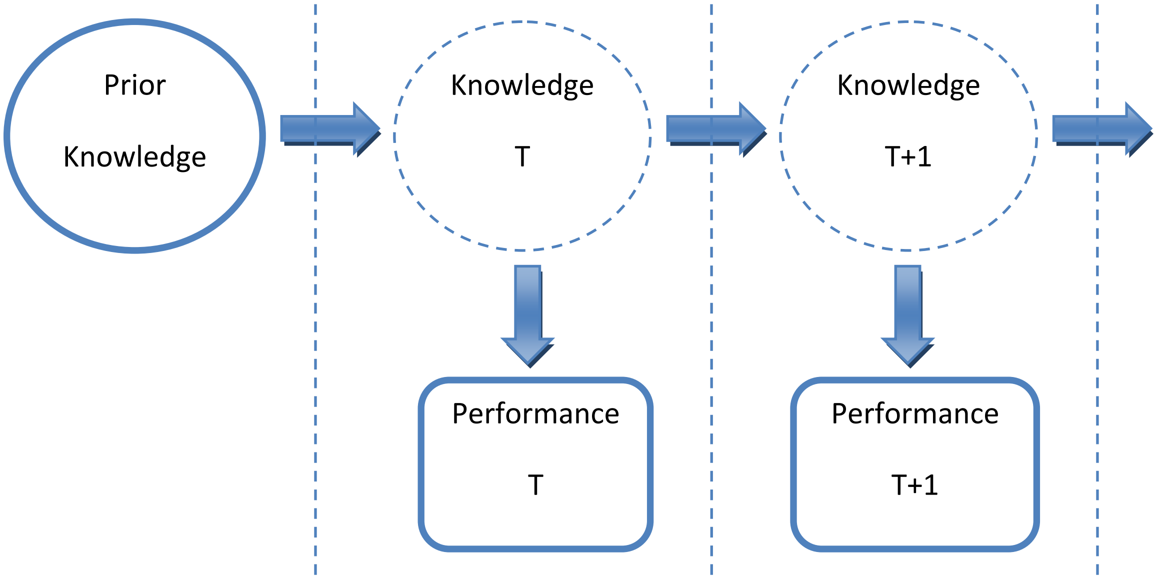


Figure 7: Dynamic Bayesian Network used for Knowledge Tracing.

Figure 7 shows a segment of a Dynamic Bayesian Network used for knowledge tracing. The first two problem slices are visible, but the network continues on to the right for as many problems as the student answers. Each arrow in the diagram represents a probabilistic relationship. Some typical parameters in detail:

“lean”	P(Knowledge T+1 | ¬Knowledge T)
“forget”	P(¬Knowledge T+1 | Knowledge T)
“guess”	P(Correct | ¬Knowledge)
“slip”	P(Wrong | Knowledge)

Figure 8: Parameters of interest in Knowledge Tracing.

Here we are most interested in the “learn” parameter which represents the probability that the student will learn the skill after any one problem. A higher learn parameter thus reflects a better chance to learn from the problems. The guess and slip parameters represent the chance the student will get a problem correct when they don’t know the skill, or when they get a problem wrong even if they do know the skill, respectively. The forget parameter is worth mentioning because by convention it is set to zero. That is, students don’t forget skills once they learn them. There are many reasons the community chooses to ignore forgetting in the model (including limiting the parameter search space), but using a short period of data like in this study makes it easy to justify ignoring forgetting.

6. Data

As of the time of this writing data has been collected for the Effort Bar condition with the following breakdown:

- 6 classes,
- 262 students,
- Almost 8000 problems completed.

As control for this group (extinction data has not been collected yet) problem logs were collected from those same student/class groupings from before the Effort Bar was activated.

7. Results

Using the data obtained thus far I ran the logs through the offline gaming detector as well as built Knowledge Tracing models for training to try to show learning gains.

7.1 Effort Bar Gaming Results

Running the offline gaming detector on the problem logs yields a list of user ids along with the total number of problems completed, the number of problems they gamed on, and the total number of gaming events for that user. I look at two metrics of gaming: gaming events per problem and percent of problems gamed. The first is concerned with the absolute number of actions the student performs that are classified as gaming. The second measure can be more forgiving since it only looks at the percent of problems where at least one gaming event occurred. This allows for the possibility that a student may perform many gaming actions on a single problem. It allows us to distinguish students who may have moments of gaming but are on the most part working hard from those who game often (on many problems).

We first look at gaming events per problem using all the data in Figure 9. Although the Effort Bar data has a higher mean number of gaming events, there is too much uncertainty to decide the true relationship. All error bars show 95% confidence.

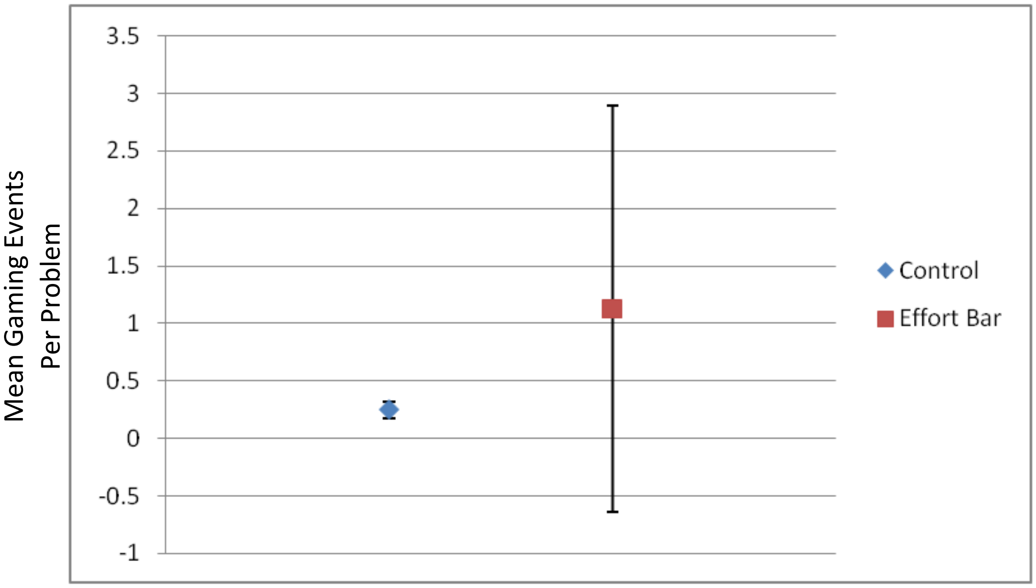


Figure 9: Gaming events per problem (full data).

Next, we look at the overall percent of problems on which the student gamed. Although this is not a paired comparison there is a clear relationship showing reliably that students in the effort bar group gamed on a smaller percentage of their total problems.

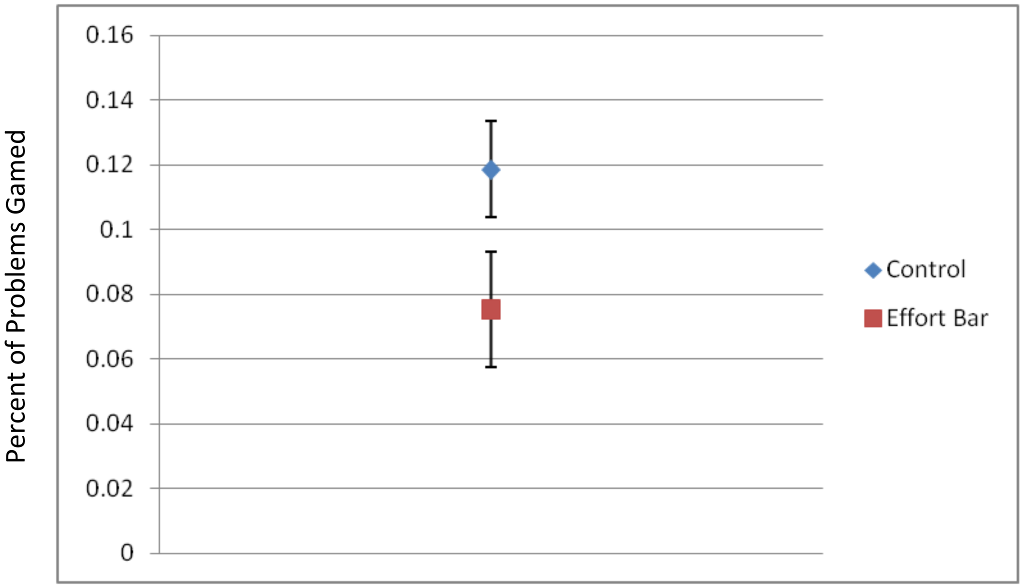


Figure 10: Percent of problems gamed (full data).

However, looking at the full data may include users in one group that don’t have data in the other. Also it only provides a coarse aggregate view of the groups gaming activity. It would be useful to know how much each individual student changed in their gaming practices. I pruned the data such that only matched users remained (only users who had logs in both groups were included). In addition to matching users I pruned out the students who completed few problems (less than 10 complete problems).

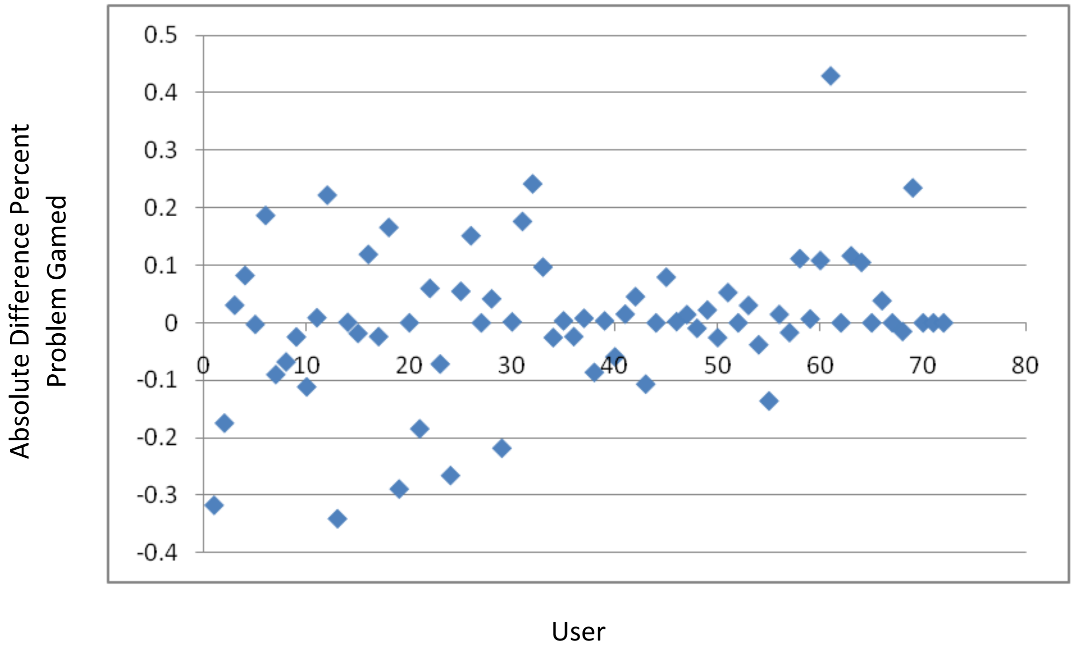


Figure 11: Difference in percent problems gamed (matched and pruned).

There was no significantly reliably result for the difference in percent problems gamed using the matched pruned data. It is possible that the relationship we saw in the aggregate data was the result of some of the students in the control group not finishing enough of the problems once the Effort Bar was activated. There was an almost significant result for the difference in gaming events per problem (p=0.21). However, as we can see from the graph below, this was likely due to the presence of two outliers. Ignoring these two points the rest of the graph is centered around zero, pointing to no difference between the two groups.

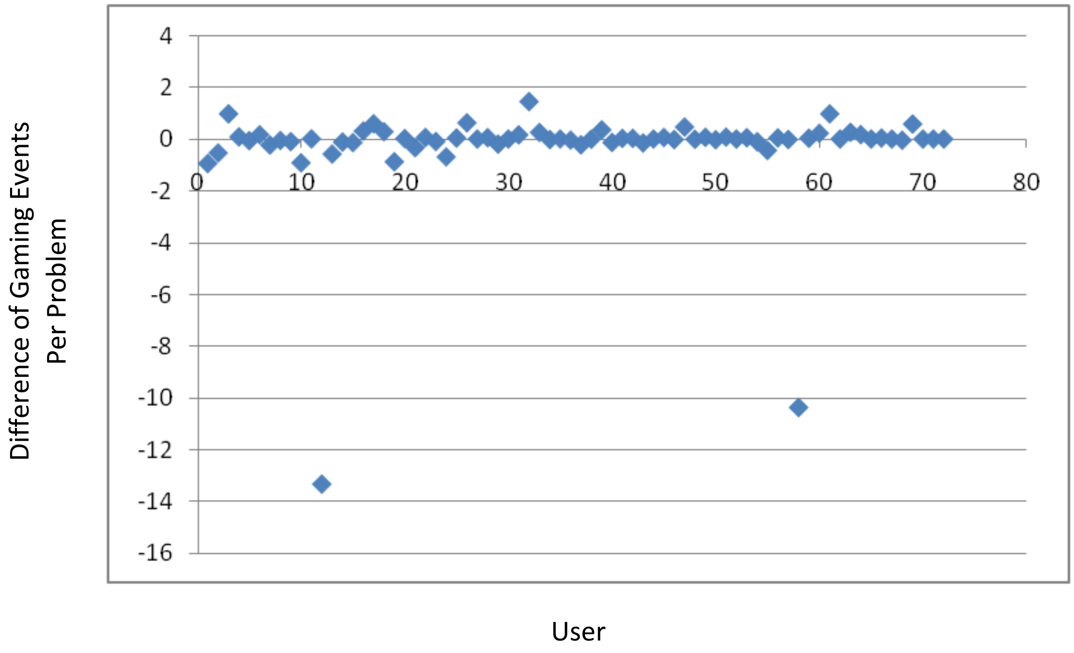


Figure 12: Difference in gaming events per problem (matched and pruned).

7.2 Effort Bar Learning Results

Knowledge Tracing models were built and trained on the Effort Bar data using the freely available matlab package BNT-SM [6]. The package is designed specifically for building and training dynamic Bayesian networks for student modeling. As such, no major modifications were required to build the Knowledge Tracing models. There are two prevailing ways of splitting up the data into models. The first is to model how all students learn specific skills, the skill model. The second is to model how each individual student learns all skills, the user model. All p-values are two-tailed.

7.2.1 Skill Models

I built two skill models, one on the data before the Effort Bar was activated, and the other on data after it was activated. I then compared the difference in the probabilistic parameters that are return by the training process. Specifically I was interested in the learn parameter, so all graphs will be of this parameter. In the skill models there is a learn parameter for every skill.

In figure 13 below we can see the difference in learn parameter (Effort Bar – Control) for all the trained skills. We see most of the skills remain roughly the same, however the average gain in learning rate is 0.065, that is, almost a 7% greater chance to learn from a problem. This result is only marginally reliable however (p=0.64).

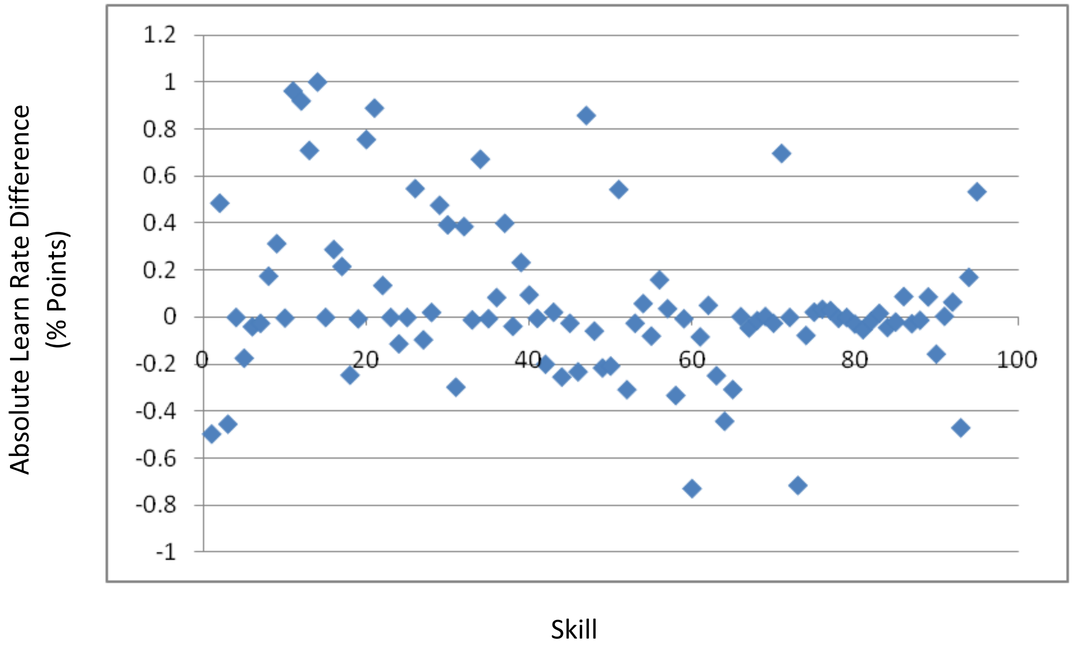


Figure 13: Learning rate difference by skill (non-pruned)

In order to reduce the likelihood that this result was due to small amounts of training data for some skills that act as outliers I pruned the data sets. Any skills with fewer than 20 problems completed were removed from the training data. In figure 14 below we can see the results of training on the pruned data. Training only on the skills with more data (that are likely a better estimate of the true value for that skill) are results actually improve. The mean learning gain per skill in the Effort Bar group is 0.082, over an 8% learning gain. This result is also slightly more reliable with p=0.52.

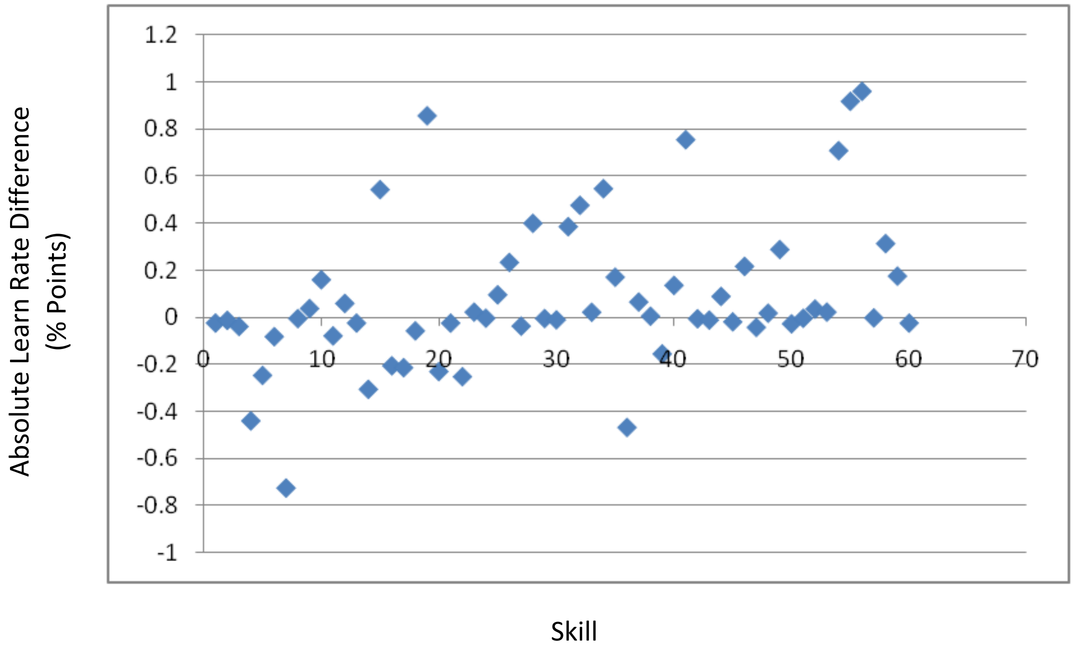


Figure 14: Learning rate difference by skill (pruned)

7.2.2 User Models

The results for the skill models are encouraging and while it is nice to be able to say that students could learn certain skills better, we would like to be able to say something about whether or not individual students were learning better. I pruned out any users with fewer than 20 problem responses. The graph in figure 15 bellow shows the aggregate results for the mean learn rate for the user models from each group.

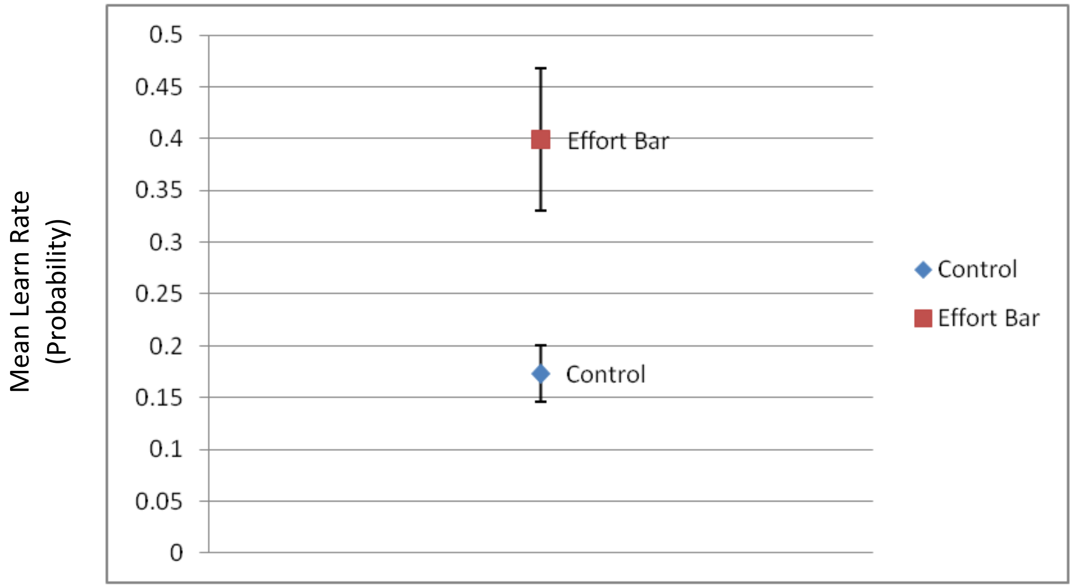


Figure 15: Mean learn rate by condition (pruned, non-matched)

As we can see the Effort Bar group seems to reliably outperform the control group by over 10 percent! However, if we match users (removing any users that don't have logs in both groups) we get the following graph in figure 16. While it is still clearly a positive result for the Effort Bar, the fact that so many points are at or very close to 1 is disturbing. This would mean that those students have a probability of learn that is near unity in the Effort Bar group and near zero in the control. Due to this I decided to prune these results to only students with at least 20 problems completed.

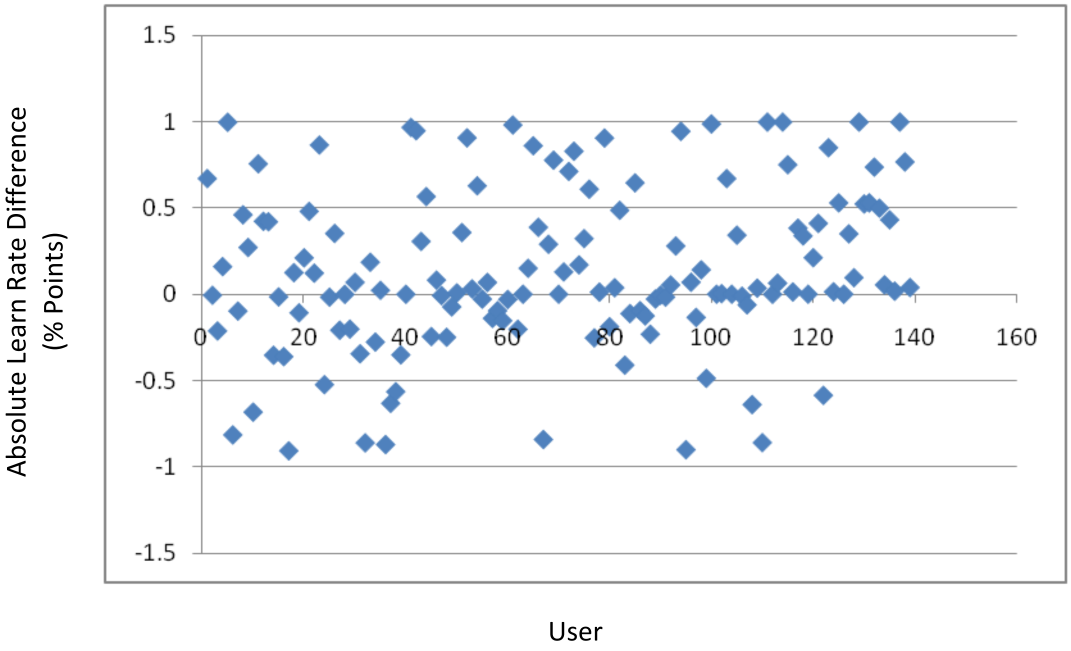


Figure 16: Learning rate difference by user (matched, non-pruned)

The results of the pruned data is in figure 17 below. Unfortunately the learning gain no longer exists (mean gain 1.2%, p=0.846). We have less suspicious looking data now, however that doesn't mean the original result was spurious. It is possible that some students actually did have a much higher chance of learning in the Effort Bar group than control (gamers perhaps) but they did not complete enough problems to make it past pruning. This hypothesis is unlikely, however it doesn't change the fact that there was a learning gain in the aggregate data.

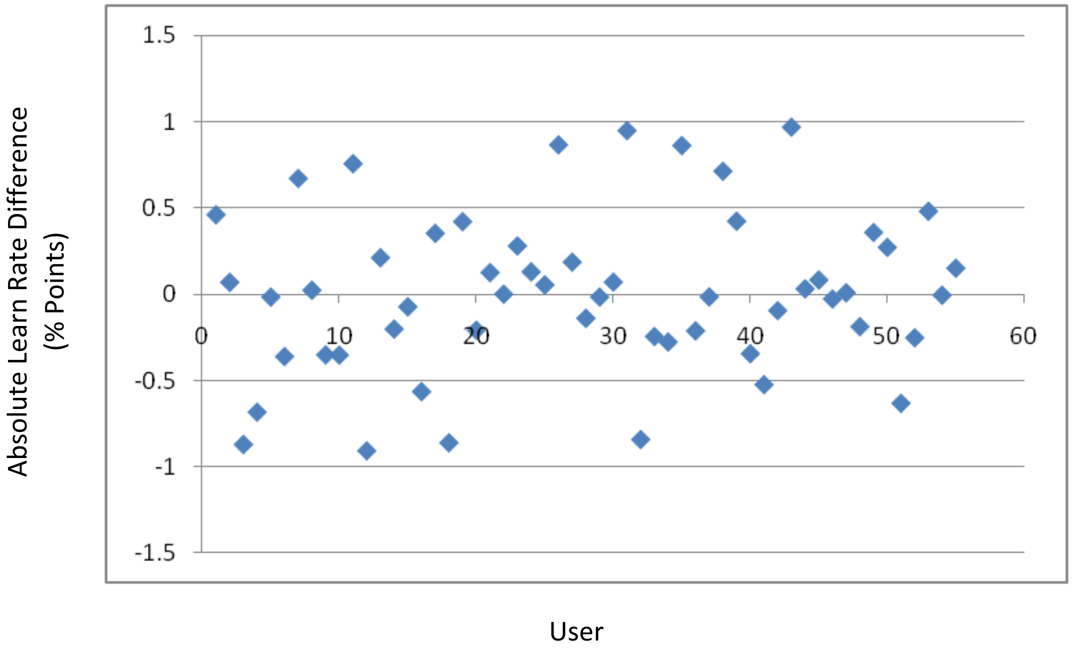


Figure 17: Learning rate difference by user (matched, pruned)

8. Conclusion

We were unable to show a definite positive reliable effort for reducing gaming or improving learning the aggregate results did show a reliable reduction of gaming behavior and increase in learning. This could suggest that more data is needed to show a definite effect. This is especially likely considering the fact that gaming behavior makes up such a small portion of overall student actions. I have build a heuristic detector for gaming behavior both offline and as an integrated part of the ASSISTments system. It is my hope that the online version especially will be useful to future researchers. The fact that it makes gaming data easily available should make future gaming research (or interventions) easier with a lower barrier to entry. I have built two separate interventions for preventing gaming behavior in students and completed an initial evaluation of the Effort Bar.

The Effort Bar is in daily use in one school now and the teachers and students have commented positively on its presence.

References

1 Baker, R.S., Corbett, A.T., Koedinger, K.R. (2004) *Detecting Student Misuse of Intelligent Tutoring Systems* . Proceedings of the 7th International Conference on Intelligent Tutoring Systems, 531-540.

2 Baker, R.S., Corbett, A.T., Koedinger, K.R., Wagner, A.Z. (2004) *Off-Task Behavior in the Cognitive Tutor Classroom: When Students "Game The System"*. Proceedings of ACM CHI 2004: Computer-Human Interaction, 383-390.

3 Baker, R.S.J.d., Corbett, A.T., Koedinger, K.R., Evenson, E., Roll, I., Wagner, A.Z., Naim, M., Raspat, J., Baker, D.J., Beck, J. (2006) *Adapting to When Students Game an Intelligent Tutoring System*. Proceedings of the 8th International Conference on Intelligent Tutoring Systems, 392-401.

4 Corbett, A.T., Anderson, J.R. (1995) *Knowledge Tracing: Modeling the Acquisition of Procedural Knowledge.* User Modeling and User-Adapted Interaction, 4: 253-278.

5 Gong, Y., Beck, J., Heffernan, N. & Forbes-Summers, E. (2010) *The impact of gaming (?) on learning at the fine-grained level.* International Conference of Intelligent Tutoring Systems.

6 Project Listen. *BNT-SM: Bayes Net Toolkit for Student Modeling.* Carnegie Mellon University. http://www.cs.cmu.edu/~listen/BNT-SM/ (accessed March 2010).

7 Shih, B., Koedinger, K.R., and Scheines, R. (2008). *A response time model for bottom-out hints as worked examples.* In Baker, R.S.J.d., Barnes, T., & Beck, J.E. (Eds.) *Proceedings of the First International Conference on Educational Data Mining, 117-126*.

8 *Walonoski, J.* & Heffernan, N.T. (2006a). *Detection and analysis of off-task gaming behavior in intelligent tutoring systems.* In Ikeda, Ashley & Chan (Eds.). Proceedings of the Eight International Conference on Intelligent Tutoring Systems. Springer-Verlag: Berlin. pp. 382-391. 2006.

9 Walonoski, J. & Heffernan, N. (2006b). *Prevention of off-task gaming behavior in intelligent tutoring systems.* In Ikeda, Ashley & Chan (Eds.). Proceedings of the 8th International Conference on Intelligent Tutoring Systems. 2006, LNCS 4053. Springer-Verlag: Berlin. pp. 722-724.

Appendix A. Additional Details of Effort Bar Implementation.

The original version of the Effort Bar had a bulky remoting interface that allowed communication between the browser and the flash object. Also the flash object required a HaXe client-side Javascript library to be included. This greatly increased the amount the client had to download when loading the Effort Bar and it required the bar to be in an embedded frame on the page to prevent conflicts with other Javascript libraries on the page. The Effort Bar was upgraded to use a new version of remoting which greatly simplified browser interaction. I also modified the bar so that dependencies to the HaXe javascript library were removed. This reduced the content the client has to download and allowed the bar to be directly embedded into the page.

During an initial test of the Effort Bar some computers were unable to load the bar and this caused problems with the client engine. Many of the computers and software used in schools are old and not upgraded. A version checking script was added to make sure that the computer had appropriate versions of the browser and flash plugin and would refuse to load the Effort Bar if it didn’t.

The Effort Bar was added as a “tutor setting” which allows teachers to turn it on or off on a class by class basis. Hover text was added to the points on the bar when the mouse was moved over them. The client engine was modified to pass information about the student’s answers to the bar. This allows teachers to view student’s previously submitted answers by mousing-over those answer points on the bar.

Several visual changes were made to the bar since the original version. There were several requests to reduce the size of the bar. Over a series of updates the vertical size of the bar was reduced so that it took up less room on the student’s screen. The original version of the bar had a background display of student “effort” which was a crude heuristic based on student performance. Because this information wasn’t particularly useful considering the information already presented by the bar, and the fact that the background display contributed to the bar being visually overwhelming, this feature was disabled.

More recently a user/developer stress test and evaluation was conducted for the Effort Bar. A few minor bugs were uncovered. One problem was that some monitors had poor color resolution and it was difficult to see the colored backgrounds on the bar; this suggested the color saturation of the background should be increased. In addition there was some confusion over the vertical change in points and it was suggested that it be removed and replaced with a 3-level design where hints, correct answers, and incorrect answers each occupied a separate vertical position.

Appendix B. Additional Features Not Used in This Study.

Student Model

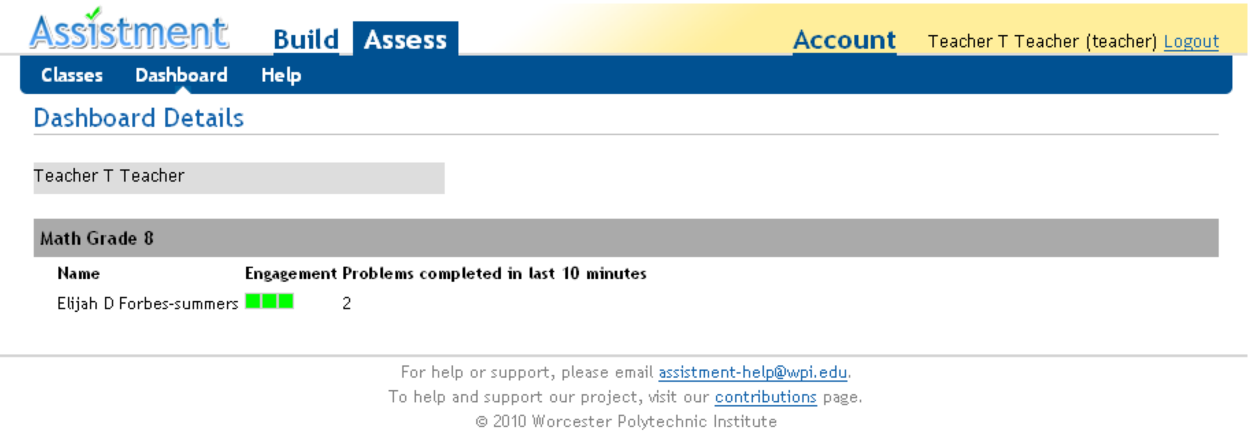
At one point we had considered added a “Student Model” feature to the system. It would be a place where developers working on other features could store persistent information about students. A prototype of this feature was made, but it was woefully underspecified. A previous modification had already been made to the “progress” feature of the tutor to allow for any types of variables to be added into the progress, which was stored per student. This modification allowed for persistent per student information required by the gaming detector to be stored. In addition other developers could add their own variables to be stored easily. Because there was no outside need at the time for a separate user model feature, and because the progress modification had already been made, the prototype was discontinued and the feature was dropped.

Experiment Management Feature

Neil requested that I make an experiment management feature for assistment. This feature would allow a researcher to define experimental groups, handle applying experimental conditions to certain groups, provide a simple way to view the data from those groups, and keep track of which students were in which experiments. I spent a week or two looking into this and designing modifications to the assistment system. However, the types of conditions used in studies vary dramatically and there are already fairly useful ways of handling assignment of these conditions (assigning students to two different sections and assigning separate content to each, or random selection within problem sets. Also the tutor settings interface allows conditions on a class basis). The benefit of such a system is far outweighed by the effort required to effectively implement it. The project was dropped.

Teacher Dashboard

This feature would provide a screen for teachers to monitor the progress of an entire class all at once. It would also allow teachers to indicate to students that they are being watched. I built a prototype page with very simple information on what students in a class were doing and the ability to "ping" students, displaying an eye icon on their screen if the teacher wanted them to see that she was watching what they were doing. I planned to make some sort of effort bar-like display for each student so teacher could see what they were doing. It was requested that this screen, instead of being a separate page for the teacher, was a persistent across the entire website while the teacher was logged in (and had a class working) and took up a small section of the screen. This required that only the most salient information be displayed at any time (how many students working, alerts to any students who might not be working hard, etc.). However, this feature encouraged teachers to be at their desk rather than walking amongst students helping them. Because of this the feature was dropped.



Dashboard Details

Teacher T Teacher

Math Grade 8

Name	Engagement	Problems completed in last 10 minutes
Elijah D Forbes-summers	🟩🟩🟩	2

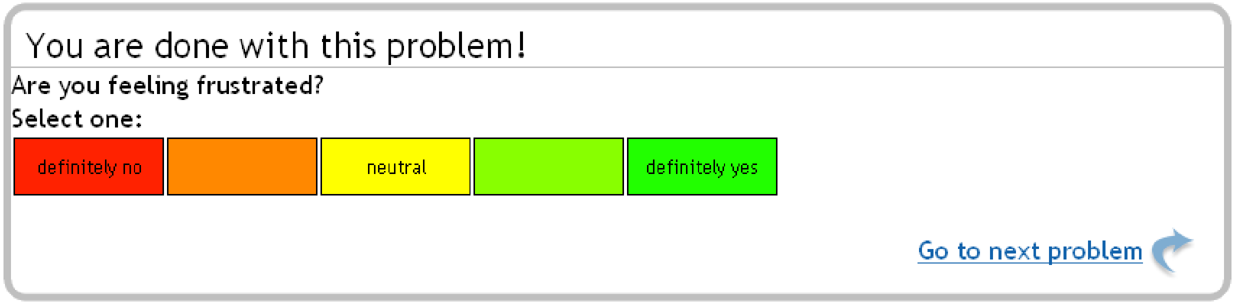
For help or support, please email assistment-help@wpi.edu.
To help and support our project, visit our contributions page.
© 2010 Worcester Polytechnic Institute

Figure 18: Prototype of the teacher dashboard.

Survey System

A consultant to the project suggested that we obtain survey ratings of how students react to the interventions. Because we were assigning conditions remotely it was decided that the surveys should be built into assistment. Rather than making a survey assignment the surveys would be randomly asked while the student was doing their normal assignments. In this way we would have answers to questions over time hopefully presenting a better picture of how the students felt and we wouldn't have to request that teachers assign a separate survey.

A prototype survey system was built that randomly choose to ask a question from a javascript library and recorded the answer within the actions log of the current problem. This system had support for scale and multiple choice questions and automatically handled the GUI generation and logging functions. Students were not required to answer the survey question in order to move on to prevent the survey from interfering with their work. It was requested that this be built into a full feature of the system. I spent about a week laying out the design for such a system before the project was dropped when it became apparent that a full system would be significantly more work.

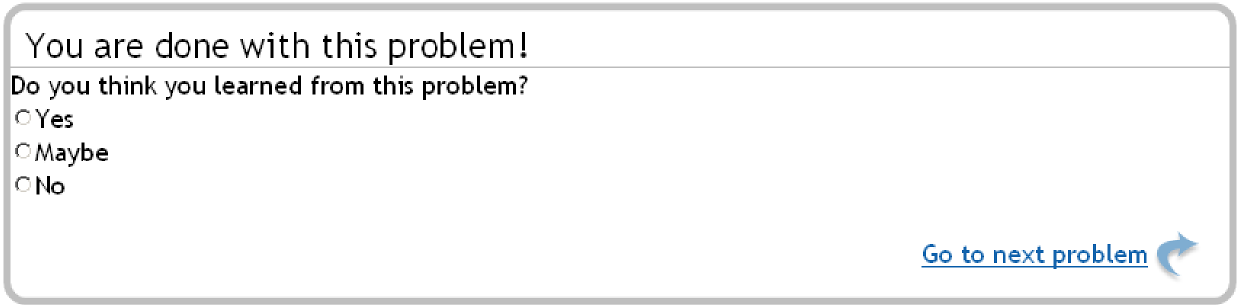


You are done with this problem!

Are you feeling frustrated?

Select one:

Figure 19: Sample scale-type survey question.



You are done with this problem!

Do you think you learned from this problem?

○ Yes
○ Maybe
○ No

Figure 20: Sample multiple-choice type survey question.

Second Attempt Before Scaffolding

When we were discussing rewarding students for working hard, one of the possibilities was to give them an additional attempt to answer a main problem before they were forced into scaffold problems. Main problems with scaffolding in the ASSISTments system normally only give a student one try to get the problem correct before they have to complete all the scaffold questions. Sometimes a student will make a small mistake on the main problem, but still know how to solve the problem. In these cases the student has to complete all the scaffold problems and it understandably frustrated. The possibility of making an additional attempt before starting scaffolding alleviates this problem. This feature was implemented but not used in the operant conditioning intervention.

Retry Main Problem

This feature is similar to the “Second Attempt Before Scaffolding” feature described above. The difference is that this feature would allow students to return to the main problem after they had already started scaffold problems and make another attempt. This is a superior feature because many times students discover how to solve a problem only partway through the scaffold problems and could prove this and move on much earlier. However, this feature also requires much deeper modifications to the ASSISTments system. The path that students take through the tutoring content is an assumed constant throughout the system and it was unclear which parts of the system would even need to be modified in order to assure their continued functioning once this change was implemented. In order to attempt to discover what changes would need to be made I hacked the client side engine so that students could at any point return to the main problem. This was accomplished by remembering main problems as the students passed them. Then when the student requested to go back to the main problem, the remembered reference was pushed back onto the problem stack. The GUI elements for the current problem were disabled and those for the main problem were re-enabled. However this caused problems with logging and server communication. In addition the client became confused if the student left the problem set and resumed it later. Normally the client “replays” all the student actions in a problem when they resume so that it looks like they are at the same point where they left off. However, with the modification the tutor would either have to return to the current problem (not rather than the main problem) or it would think the student had finished the current problem when they hadn’t.

There are a few different solutions to this problem.

- Appending actions after retry to the original problem log. This requires significant server logging modification and might disrupt reports.

- Creating a new copy of the original problem. This disrupts the replay function, and because the current problem isn’t properly terminated requires deeper modification of the client engine.
- Creating a new action for terminating current problems that signals that the student returned to the main problem. This solves replay problems, but there are still report problems. It also provides a way of limiting the number of times students can retry the main problem, which was a problem in other solutions (students could bounce between main and current problems as much as they wanted in order to keep retrying the problem).

Because of the significant modifications required to implement this feature we decided to focus on the “Second Attempt Before Scaffolding” feature.

Appendix C. Offline Detector Used For This Study [Ruby]

```
require 'postgres-pr/connection'

Consecutive_Bottom_Out_Hint_Limit = 3
Consecutive_Rapid_Guess_Limit = 2
Gaming_Maximum = 1
Gaming_Minimum = 0
Positive_Increment = 0.5
Reading_Rate = 400

in_file = File.new("[input problem_log_ids]", "r")
out_file = File.new("[output_file]", "w")

bodies = Hash.new(nil)
hints = Hash.new(nil)

$fixed_reading_rate = Reading_Rate / 60000.0

# Per student gaming events.
gaming_events = Hash.new(0)

total_problems = Hash.new(0)
gamed_problems = Hash.new(0)

current_user = nil
current_problem = nil
hint_count = 0
```

```
consecutive_BOH_count = 0
did_game = false
thinking_primed = false
thinking_gamed_count = 0
last_action = nil
last_action_time = 0


def readingTime text
    if text.nil?
        return 0
    else
        stripped_text = text.gsub(/<[^>]*>/, '')
        stripped_text = stripped_text.gsub(/&[^;|^&]*;/, '')
        stripped_text = stripped_text.gsub(/\s+/, ' ')
        letters = stripped_text.split(//)
        return ((letters.length / 5) / $fixed_reading_rate)
    end
end


conn = PostgresPR::Connection.new('[database]', '[user]', '[password]')

#
# Get list of problem_log_ids that we want to analyze.
#

puts "Reading in problem_ids"
problem_logs = in_file.readlines



#
# Get problem and hint text for the included problems.
#

puts "Fetching problem body texts from database"
body_results = conn.query("select id, body from problems
                            where id IN (
                                select distinct problem_id from
problem_logs as pl
                                where pl.id IN (
                                    #{problem_logs.join(",")}
                                )
                            )")
#

body_results.rows.each do |body|
    bodies[body[0].to_i] = body[1]
```

```
end

puts "Fetching problem hint texts from database"
hints_results = conn.query("select problem_id, hints.value from tutor_strategies as ts
                                    join hints on
hints.tutor_strategy_id = ts.id
                                    where problem_id IN (
                                        select distinct
problem_id from problem_logs as pl
                                        where pl.id IN (

    #{problem_logs.join(",")}
                                        )
                                    )
                                    order by problem_id,
hints.position")
#

temp_prob_id = nil
hint_array = nil
hints_results.rows.each do |hint_info|
    if hint_info[0] == temp_prob_id
        hint_array.push hint_info[1]
    else
        if !hint_array.nil?
            hints[temp_prob_id.to_i] = hint_array
        end
        hint_array = Array.new()
        hint_array.push hint_info[1]
    end
    temp_prob_id = hint_info[0]
end
hints[temp_prob_id.to_i] = hint_array


#
# Now get the problem logs.
#

puts "Fetching problem_logs from database"
results = conn.query("select user_id, problem_id, actions, assignment_id from problem_logs
                            where id IN (
                                #{problem_logs.join(",")}
                            )
                            order by user_id, start_time")

#
```

```
puts "Processing Logs..."
results.rows.each do |action_log|
    current_problem = action_log[1].to_i
    hints_array = nil
    body_text = nil

    #If new user, reset
    unless (action_log[0] == current_user)
        hint_count = 0
        consecutive_BOH_count = 0
        consecutive_rapid_guess_count = 0
        last_action = nil
        thinking_primed = false
        did_game = false
        thinking_gamed_count = 0
        last_action_time = 0
    end
    current_user = action_log[0]
    total_problems[current_user.to_i] += 1

    actions = YAML::load action_log[2]
    actions.each do |action|
        if (action[0] == "start")
            if (action[1] - last_action_time) > 3600000
                consecutive_BOH_count = 0
                consecutive_rapid_guess_count = 0
                thinking_primed = false
            end
            body_text = bodies[current_problem]
            hints_array = hints[current_problem]
            last_action_time = action[1]

        elsif (action[0] == "resume")
            if (action[1] - last_action_time) > 3600000
                # Reset BOH and thinking
                consecutive_BOH_count = 0
                consecutive_rapid_guess_count = 0
                thinking_primed = false
            end
            last_action_time = action[1]

        elsif (action[0] == "hint")
            # Check for fast response.
            min_time = 0
            if last_action == "hint"
                min_time = readingTime(hints_array[hint_count])
            elsif last_action == "start"
                min_time = readingTime(body_text)
            end
            if action[1] < min_time
                did_game = true
                gaming_events[current_user.to_i] += 1
                thinking_gamed_count += 1
                #puts "fast hint request"
            end

            hint_count += 1

            #check for BOH
            unless hints_array.nil?
            if hint_count >= hints_array.length
                consecutive_BOH_count += 1
                thinking_primed = true
                if consecutive_BOH_count >= Consecutive_Bottom_Out_Hint_Limit
                    did_game = true
                    gaming_events[current_user.to_i] += 1
                    #puts "BOH limit"
                end
            end
            end

            last_action_time += action[1]

        elsif (action[0] == "scaffold")
            min_time = 0
            if last_action == "hint"
                min_time = readingTime(hints_array[hint_count])
            elsif last_action == "start"
                min_time = readingTime(body_text)
            end
            if action[1] < min_time
                did_game = true
                gaming_events[current_user.to_i] += 1
                #puts "fast scaffold"
            end

            last_action_time += action[1]

        elsif (action[0] == "answer")
            # Check for thinking
            if thinking_primed && last_action == "hint" && action[1] > 10000
                consecutive_BOH_count = 0
                gaming_events[current_user.to_i] -= thinking_gamed_count
                if gaming_events[current_user.to_i] == 0
                    did_game = false
                end
                #puts "thinking!"
            end

            # Check for rapid guess
            if last_action == "answer" && action[1] < 2000
                consecutive_rapid_guess_count += 1
                if consecutive_rapid_guess_count >= Consecutive_Rapid_Guess_Limit
                    did_game = true
                    gaming_events[current_user.to_i] += 1
                    #puts "rapid guess"
                end
            else
                consecutive_rapid_guess_count = 0
            end

            # Check for fast response.
            min_time = 0
            if last_action == "hint"
                min_time = readingTime(hints_array[hint_count])
            elsif last_action == "start"
                min_time = readingTime(body_text)
            end
            if action[1] < min_time
                did_game = true
                gaming_events[current_user.to_i] += 1
            end

            last_action_time += action[1]

        elsif (action[0] == "end")
            unless hints_array.nil?
                if hint_count < hints_array.length
                    consecutive_BOH_count = 0
                end
            end

            if did_game
                gamed_problems[current_user.to_i] += 1
            end

            hint_count = 0
            hint_count = 0
            consecutive_rapid_guess_count = 0
            last_action = nil
            thinking_primed = false
            did_game = false
            thinking_gamed_count = 0
            body_text = nil
```

```
            hints_array = nil
        end

        last_action = action[0]
    end
end

out_file.puts "user, total_problems, gamed_problems, gaming_events"
    total_problems.each_key do |key|
        out_file.puts key.to_s + "," + total_problems[key].to_s + "," +
gamed_problems[key].to_s + "," + gaming_events[key].to_s
    end

conn.close
out_file.close
```

Appendix D. Online Gaming Detector (Standalone engine) [JavaScript]

```
/**

Some variables that people might want to change at some point

**/

var CONSECUTIVE_BOTTOM_OUT_HINT_LIMIT = 3;
var CONSECUTIVE_RAPID_GUESS_LIMIT = 2;
// Generally, gaming = 1, not gaming = 0
var GAMING_MINIMUM = 0;
var GAMING_MAXIMUM = 1;
// Ammount the gaming value decreases when the student does something good (not gaming on a problem).
var POSITIVE_INCREMENT = 0.5;
// Reading rate in words per minute
// This is fast because we don't want to penalize fast readers on accident
var READING_RATE = 400;

/**

End of those variables that people might want to change at some point

**/

//var EngagementMonitor = false;
var prevGaming = 0;

function updateEngagementDisplay(gaming) {
  img = 'green-circle.png';
        if (gaming > 0 && gaming < 0.5) {
                img = 'yellow-circle.png';
        } else {
                img = 'red-circle.png';
        }

        $('engagement').innerHTML = '<img src="../../../images/' + img + '">';
        if (gaming != prevGaming) {
                if (gaming == 0) {
                        //alert("good");
                } else if (gaming == 1) {
                        //alert("bad");
                }
        }
        prevGaming = gaming;
}
```

```
function GamingMonitor() {
	var gaming = 0;
	var reason = null;
	var gaming_saved = 0;
	var did_game = false;
	var thinking_primed = false;
	var hint_count = 0;
	var consecutive_BOH = 0;
	var BOH_limit = CONSECUTIVE_BOTTOM_OUT_HINT_LIMIT;
	var consecutive_rapid_guess = 0;
	var rapid_guess_limit = CONSECUTIVE_RAPID_GUESS_LIMIT;
	var last_action_time = 0;
	var last_action;
	var body = false;
	var hints = false;
	var gaming_min = GAMING_MINIMUM;
	var gaming_max = GAMING_MAXIMUM;
	var positive_increment = POSITIVE_INCREMENT;
	var reading_rate = READING_RATE;
	//convert to words per millisecond
	var fixed_reading_rate = (reading_rate / 60000);
	var log_action = false;
	var first_run = true;
	var started = false;
	var reset_BOH = false;

	this.hasLogAction = function() {
		return (log_action != false);
	};

	this.getLogAction = function() {
		temp_action = log_action;
		log_action = false;
		return temp_action;
	};

	this.getGaming = function() {
		return gaming;
	};

	this.getReason = function() {
		return reason;
	};

	this.resetLog = function() {
		log_action = false;
	};
```

```
var setGaming = function(value, why) {
    reason = why;
    if (gaming != value) {
        gaming = value;
        logGaming();
    }
};

var setSavePoint = function() {
    gaming_saved = gaming;
};

var restoreSavePoint = function() {
    gaming = gaming_saved;
    logGaming();
};

var logGaming = function() {
    log_action = new Array('gaming', 0, gaming);
};

var copyHints = function(tutor_hints) {
    var newhints = new Array(tutor_hints.length);
    for (var i = 0; i < newhints.length; i++) {
        newhints[i] = tutor_hints[i].value;
    }
    return newhints
};

var positiveAction = function() {
    reason = null;
    if (gaming != gaming_min) {
        if (gaming > (gaming_min + positive_increment)) {
            gaming = gaming - positive_increment
        } else {
            gaming = gaming_min;
        }

        // Only log a positive event if it changes the student's gaming value
        logGaming();
    }
};

var negativeAction = function(why) {
    reason = why;
    //alert(why);
    if (gaming != gaming_max) {
        gaming = gaming_max;
    }
    // always log when the student games
    logGaming();
};

this.parsePastProblem = function(problem) {
    // no need to parse if we have started getting realtime actions.
    if (started) {
        return;
    }
    var has_scaffold = false;
    // We get problem body and hints for free!
    body = problem.body;
    hints = copyHints(problem.hints);
    for (var i=0; i<problem.actions.length; i++) {
        // Process the action
        //can't reference 'this' inside a 'private' function, so we call to gm
        gm.action(problem.actions[i]);
        if (problem.actions[i][0] == "scaffold") {
            has_scaffold = true;
        }
    }
    // If the problem went to scaffolding process the actions in the scaffold problems.
    if (has_scaffold) {
        for (var j=0; j<problem.scaffolds.length; j++) {
            this.parsePastProblem(problem.scaffolds[j]);
        }
    }
};

//debugging purposes
this.printHints = function(hints) {
    out("===printing hints===");
    for (var i=0; i<hints.length; i++) {
        out(hints[i]);
    }
    out("===hints done===");
};

this.loggedAction = function(action) {
    if (!started) {
        started = true;
    }
    this.action(action);
};

this.action = function(action) {
    switch(action[0])
```

```
        {
            case 'start':
                //########## START ##########
                //log what gaming is when the problem started, just for reference when
looking in the problem_logs
                logGaming();
                body = body ||
assignment.current_assistment().current_problem().body;
                hints = hints ||
copyHints(assignment.current_assistment().current_problem().hints);
                this.printHints(hints);
                last_action_time = action[1];
                if(reset_BOH) {
                    assignment.progress_variables.BOH_count = 0;
                }
                if(first_run) {
                    consecutive_BOH = assignment.progress_variables.BOH_count;
                    first_run = false;
                }
                break;
            case 'resume':
                //########## RESUME ##########
                // Resumes are weird
                this.resetLog();
                if(first_run) {
                    consecutive_BOH = assignment.progress_variables.BOH_count;
                    first_run = false;
                }
                //if interval is greater than an hour then reset gaming
                if ((action[1] - last_action_time) > 3600000) {
                    setGaming(gaming_min, null);
                    did_game = false;
                    consecutive_BOH = 0;
                    assignment.progress_variables.BOH_count = 0;
                    // if they have been gone that long we can assume they aren't
thinking
                    thinking_primed = false;
                }
                last_action_time = action[1];
                break;
            case 'hint':
                //########## HINT ##########
                //important to check fast response before updating hint count
                //because the function only checks the previous hint (which would be
this hint if the count was incremented)
                checkFastResponse(action[1]);
                hint_count++;
                //save gaming value in case a worked example occurs later
        if (hint_count == 1) {
            setSavePoint();
        }
        //check for BOH and BOH limit
        if (hint_count >= hints.length) {
            consecutive_BOH++;
            assignment.progress_variables.BOH_count = consecutive_BOH;
            //XXX
            out(assignment.progress_variables.BOH_count);
            thinking_primed = true;
            if (consecutive_BOH >= BOH_limit) {
                negativeAction("reached the last hint too many times");
                did_game = true;
            }
        }
        last_action_time = last_action_time + action[1];
        break;
    case 'scaffold':
        //########## SCAFFOLD ##########
        checkFastResponse(action[1]);
        break;
    case 'answer':
        //########## ANSWER ##########
        //check for thinking (worked example)
        if (thinking_primed && last_action == "hint" && action[1] > 10000) {
            restoreSavePoint();
            consecutive_BOH = 0;
            assignment.progress_variables.BOH_count = 0;
        }
        //check for rapid guess
        if (last_action == "answer" && action[1] < 2000) {
            consecutive_rapid_guess++;
            if (consecutive_rapid_guess >= rapid_guess_limit) {
                negativeAction("guessed answers too often");
                did_game = true;
            }
        } else {
            consecutive_rapid_guess = 0;
        }
        checkFastResponse(action[1]);
        last_action_time = last_action_time + action[1];
        break;
    case 'end':
        //########## END ##########
        //if there was no BOH reset consecutive_BOH
        if (hints) {
            if (hint_count < hints.length) {
                consecutive_BOH = 0;
```

```
                    //alert('reseting BOH at end');
                    // Don't reset this yet, wait until we log...
                    //assignment.progress_variables.BOH_count = 0;
                    reset_BOH = true;
                }
            }
            //decrease gaming if student didn't game on this problem
            if (!did_game) {
                positiveAction();
            }
            hint_count = 0;
            body = false;
            hints = false;
            thinking_primed = false;
            did_game = false;
            break;
        default:
            break;
    }
    //record current action type
    if (action[0] != 'gaming') {
        last_action = action[0];
    }
    //XXX
    //out(consecutive_BOH + ' :: ' + assignment.progress_variables.BOH_count);
};

var readingTime = function(text) {
    // Really this is based on letters instead of words.

    if (typeof(text) == 'undefined' || text == null) { return 0; }
    // Strip html
    var stripped = text.replace(/<[^>]*>/g,'');
    // Split letters
    var letters = stripped.split(/\s*/);
    // A "word" is standardized to 5 characters
    return ((letters.length / 5) / fixed_reading_rate);
};

var checkFastResponse = function(time) {
    if (last_action == "start") {
        if (time < readingTime(body)) {
            //alert("reading time: " + readingTime(body));
            negativeAction("did not read the problem");
            did_game = true;
        }
    } else if (last_action == "hint") {
        if (time < readingTime(hints[hint_count-1])) {
```

```
                                        //alert("reading time: " + readingTime(hints[hint_count-1]));
                                        negativeAction("did not read the last hint");
                                        did_game = true;
                              }
                    }
          };
}
```

Appendix E. Effort Bar Code [HaXe]

```
class EffortBar
{

        //Width of the Graph
        static public var width : Int = 600;

        //Height of the Graph
        static public var height : Int = 50;

        //Time window to show on graph, in minutes
        static var displayTime : Int = 1;

        //Time interval for scrolling, in miliseconds
        static var zoomInterval : Float;

        //Number of vertical divisions to draw
        static var numMarkers : Int = 5;


        //Going to have to move a lot of this stuff to the main function so that
        //it responds to updates values passed from the html.

        static public var graph_width : Int;

        static var minutes : Int;
        static public var timeFrame : Float;
        static var t_interval : Float;
        static var p_interval : Float;
        static public var ttop : Float;

        static var current : Bool = true;
        static public var endTime : Float;
        static public var lastTime : Float;
        static public var firstTime : Float = 0;
        static var timer;
        static var la : flash.MovieClip;
        static var ra : flash.MovieClip;

        static var est_pt : Int;
        //static var creditlabel : flash.TextField;
        static var labels : Array<flash.TextField>;
        static var labelbg : Array<flash.TextField>;
        static var effort : Float = 50;

        static var prevtime : Float = 0;
        static public var mousex : Float;
```

```
    static public var mousey : Float;
    static var problems : Array<Problem>;
    static var problemId : Int = 50;
    //relative score (0-100)
    static var rel : Int = 50;
    static var js : haxe.remoting.Connection;


    //-------------------------------------------------------------------------
    // depth list
    // moveclip ordering
    //-------------------------------------------------------------------------
    //static public var creditlayer : flash.MovieClip =
flash.Lib.current.createEmptyMovieClip("credit", 25);
    static public var hoverlayer : flash.MovieClip = flash.Lib.current.createEmptyMovieClip("hover",
20);
    static public var pointlayer : flash.MovieClip = flash.Lib.current.createEmptyMovieClip("points",
15);
    static public var linelayer : flash.MovieClip = flash.Lib.current.createEmptyMovieClip("lines", 10);
    static public var markerlayer : flash.MovieClip =
flash.Lib.current.createEmptyMovieClip("markers", 5);



    //-------------------------------------------------------------------------
    //update
    //      input:   value    Array(String)
    //
    //      update is called by the js remoting code when a point is added.  The first
    //      element of the array is the numeric change to associate with the point
    //      (y-axis change).  The second element is the type of point:
    //  "finish_assistment", "results", "load_next", "scaffold",
    //  "incorrect_message", "hint", or "next_problem".
    //-------------------------------------------------------------------------
    //do credit changes seperately from these updates. so a point doesn't have to
    //be added to update the credit (i.e. showing what change an action will have
    //before it is performed).
    static function update(value : Array<String>)
    {
        var data = Std.parseInt(value[0]);
        var newdate = Date.now();
        prevtime = lastTime;
                //keep track of one before
        endTime = newdate.getTime();
    //update the latest time shown (visible)
        lastTime = endTime;
                    //update the latest time
        current = true;
                        //we are looking at the end of the graph
        if(firstTime == 0)
        {
            firstTime = lastTime;
                //keep track of the time first point added

                                                //does
this change for the correct point? IT SHOULDN'T!
        }
        else if(firstTime < (lastTime - timeFrame))
        {
            la._visible = true;
                    //If we have more points than we can show, activate left arrow
        }
        ra._visible = false;
                //since we are current, disable right arrow
        redraw();

        rel += data;
        if(rel > 100)
        {
            rel = 100;
        }
        if(rel < 0)
        {
            rel = 0;
        }

        //rel = Math.round((rel/100) * height);
        //var fixedpos = height - rel;
        var fixedpos = height - Math.round((rel/100) * height);

        switch(value[1])
        {
        case "finish_assistment":
            //End of Assistment (more assistments are up next in sequence)
            problems[0].addPoint(fixedpos, value[2], "correct");
            //problems[0].modifyEffort(-12);

        case "results":
                        //End of Sequence (final response before tutoring
complete)
            problems[0].addPoint(fixedpos, value[2], "correct");
            //problems[0].modifyEffort(-12);

        case "load_next":
                        //Next Assistment is loaded (click move on link)
            problems[0].close();
            problemId++;
            var temp = new Problem(problemId);
            problems.unshift(temp);
            problems[0].setCorrect(true);
            problems[1].drawBackground();

        case "scaffold":
                        //Incorrect answer or hint request which triggers a scaffold
            problems[0].setCorrect(false);
            problems[0].addPoint(fixedpos, value[2], "wrong");
            //problems[0].modifyEffort(-25);

        case "incorrect_message":
            //Wrong answer (on problem with no scaffold, or final part of scaffold)
            problems[0].setCorrect(false);
            problems[0].addPoint(fixedpos, value[2], "wrong");
            //problems[0].modifyEffort(-50);

        case "hint":
                                //Hint request and a hint sent
            problems[0].addPoint(fixedpos, value[2], "hint");
            //problems[0].modifyEffort(-25);

        case "next_problem":
                    //End of problem (more problems in assistment)
            problems[0].addPoint(fixedpos, value[2], "correct");
            //problems[0].modifyEffort(-12);
        }

        var pos = (graph_width-15) - ((EffortBar.endTime - firstTime) * EffortBar.ttop);
        pointlayer._x = pos;
        linelayer._x = pos;
}



//--------------------------------------------------------------------------
// rollOverEvent
//
//      Called when the mouse rolls over a point image.  Calls matchPointPosition
//      to determing which point is being hovered over.
//--------------------------------------------------------------------------
static public function rollOverEvent()
{
        mousex = flash.Lib.current._xmouse;
        mousey = flash.Lib.current._ymouse;
        matchPointPosition();
```

```
}



//------------------------------------------------------------------------
// rollOutEvent
//
//        Called when the mouse rolls off a point image.  Hides hover boxes.
//------------------------------------------------------------------------
static public function rollOutEvent()
{
        clearHover();
}



//------------------------------------------------------------------------
// matchPointPosition
//
//        For each problem this calls the findPoint function.
//------------------------------------------------------------------------
static function matchPointPosition()
{
        var i : Int;
        for(i in 0...problems.length)
        {
                problems[i].findPoint(mousex, mousey);
        }
}



//------------------------------------------------------------------------
// clearHover
//
//        Calls the clearHover function for each problem.
//------------------------------------------------------------------------
static function clearHover()
{
        var i : Int;
        for(i in 0...problems.length)
        {
                problems[i].clearHover();
        }
}
```

```
//--------------------------------------------------------------------------
//redraw
//
//        Call all the functions that update the visuals on the graph so things MOVE
//--------------------------------------------------------------------------
static function redraw()
{
        drawMinutemarkers();
}



//--------------------------------------------------------------------------
// getRatioArray
//        input:   effort    Float
//        output: ratios    Array(Float)
//
//        Calculates a gradient array for drawBackground based on effort
//--------------------------------------------------------------------------
static function getRatioArray(effort : Float)
{
        var ratios:Array<Float>;

        //Check for proper effort values, or correct them.
        if(effort > 100)
        {
                effort = 100;
        }
        else if(effort < 0)
        {
                effort = 0;
        }

        //The gradient works on values from 0-255, so we need to convert from the
        //0-100 effort score into that.
        var scaled:Float = (effort * 2.55);

        //set top and bottom limits, or return the scaled values.
        if(scaled < 15.9375)
        {
                ratios = [0, (scaled + 15.9375)];
        }
        else if(scaled > 239.0625)
        {
                ratios = [(scaled - 15.9375), 255];
        }
        else
        {
            ratios = [(scaled - 15.9375), (scaled + 15.9375)];
        }

        return ratios;
    }



    //--------------------------------------------------------------------------
    // drawBackground
    //      input:  effort  Float
    //
    //      Draws the yellow gradient background based on the effort (0-100) passed in.
    //--------------------------------------------------------------------------
    static function drawBackground(effort : Float)
    {
        var mc : flash.MovieClip = flash.Lib.current;
        var matrix:Dynamic = new flash.geom.Matrix();
        matrix.createGradientBox(600, 100, -1.571, 0, 0);
        mc.beginGradientFill("linear",

[0xFFEE9B, 0xFFFFFF],
                                                    [100.0,
100.0],

getRatioArray(effort),

matrix);
 mc.moveTo(0,0);
 mc.lineTo(0,100);
 mc.lineTo(600,100);
 mc.lineTo(600,0);
        mc.lineTo(0,0);
 mc.endFill();

        //var js = haxe.remoting.Connection.jsConnect();
        //js.JsMain.gotEffort.call([effort]);
    }



    //--------------------------------------------------------------------------
    // updateBackground
    //
    //      Takes a scaled average of the most recent effort scores and passes it to
    //      drawBackground.
    //--------------------------------------------------------------------------
    static public function updateBackground()
```

```
{
        var scaledEffort : Float = 0;

        if(problems.length > 2)
        {
                scaledEffort += (0.5 * problems[0].getEffort());
                scaledEffort += (0.3 * problems[1].getEffort());
                scaledEffort += (0.2 * problems[2].getEffort());
        }
        else if(problems.length > 1)
        {
                scaledEffort += (0.5 * problems[0].getEffort());
                scaledEffort += (0.5 * problems[1].getEffort());
        }
        else
        {
                scaledEffort = problems[0].getEffort();
        }

        drawBackground(scaledEffort);
}



//---------------------------------------------------------------------------
// drawCredit
//  input:  credit  Int
//
//      Updates the partial credit display with the passed in credit value.
//---------------------------------------------------------------------------
//static public function drawCredit(credit : Int)
//{
 // var pt_size = est_pt;
  //creditlabel.setNewTextFormat(new flash.TextFormat("verdana", pt_size, null, true));
  //creditlabel.text = Std.string(credit);
  //while(creditlabel.textWidth < (creditlabel._width - 2))
 // {
 //   pt_size++;
    //creditlabel.setNewTextFormat(new flash.TextFormat("verdana", pt_size, null, true));
    //creditlabel.text = Std.string(credit);
 // }
  //pt_size--;
  //creditlabel.setNewTextFormat(new flash.TextFormat("verdana", pt_size, null, true));
 // creditlabel.text = Std.string(credit);
  //scale height
  //while(creditlabel.textHeight > (creditlabel._height - 2))
  //{
  //  pt_size--;
```

```
          //  creditlabel.setNewTextFormat(new flash.TextFormat("verdana", pt_size, null, true));
          //  creditlabel.text = Std.string(credit);
         // }
        //  //center
        //  creditlabel._x = graph_width + ((creditlabel._width - creditlabel.textWidth) / 2) - 2;
        //  creditlabel._y = 0 + ((creditlabel._height - creditlabel.textHeight) / 2) - 5;
        //}



        //------------------------------------------------------------------------
        // convertTime
        //      input   seconds         Float
        //
        //      Takes a time in seconds and returns a string with formatted minutes and
        //      seconds. (e.g. 338 --> 5:38)
        //------------------------------------------------------------------------
        static public function convertTime(seconds : Float)
        {
                var remainder = seconds % 60;
                var minutes = (seconds - remainder) / 60;
                var newseconds : String;
                if(remainder < 10)
                {
                        newseconds = "0" + Std.string(remainder);               //Always display two
digits for seconds
                }
                else
                {
                        newseconds = Std.string(remainder);
                }
                var converted : String = Std.string(minutes) + ":" + newseconds;
                return converted;
        }



        //------------------------------------------------------------------------
        // drawMinutemarkers
        //
        //      This draws numMarkers vertical markers at regular intervals on the graph.
        //      It also updates labels which show what relative time the markers represent
        //------------------------------------------------------------------------
        static function drawMinutemarkers()
        {
                var mc : flash.MovieClip;
                mc = markerlayer.createEmptyMovieClip("grid", 5);
                mc.clear();
```

```
		mc.lineStyle(1.0, 0x8888FF, 100.0);
		var offset : Float = 7.5;   //middle of points
		var timeBase : Float = 0;
		var laps : Float;


		if(!current)
		{
			laps = ((lastTime - endTime) % t_interval);

													//Offset
the start of the marker based on how far we scrolled
			offset = ((laps / timeFrame) * (graph_width-15)) + 7.5;

			//TimeBase keeps track of how many marker divisions are hidden on the right
(i.e. how far back in time we are looking)
			timeBase = (((lastTime - endTime) - laps) / t_interval);
		}


		var i : Int;
		for(i in 0...(numMarkers+1))
		{
			mc.moveTo((i*p_interval) + offset, 0);				//draw markers
			mc.lineTo((i*p_interval) + offset, height);

			labels[i]._x = ((i*p_interval) + offset);			//move labels to line up with
markers and update their text
			labelbg[i]._x = labels[i]._x - 2;
			labels[i].text = "-" + convertTime((numMarkers - i + timeBase) *
(t_interval/1000));

			if(current && (i == numMarkers))
			{
				labels[i].text = "";
				//don't display anything for the right-most ("now") label
			}

			labelbg[i].text = labels[i].text;
		}
	}



	//---------------------------------------------------------------------------
	// drawArrows
	//
	//	Create the arrow images and some functions to handle their events.  These
```

```
    //      are for scrolling the graph view when there are more points than fit into
    //      the display.
    //--------------------------------------------------------------------------
    static function drawArrows()
    {

    //create left arrow
        la = flash.Lib.current.attachMovie("larrow", "left-arrow", 1000);
        la._x = 0;
        la._y = 0;
        la._height = height;
        la._visible = false;


    //create right arrow
        ra = flash.Lib.current.attachMovie("rarrow", "right-arrow", 1001);
        ra._x = graph_width - 15;
        ra._y = 0;
        ra._height = height;
        ra._visible = false;

        la.onPress = function()
            //When someone clicks the left arrow...
        {
            timer = new haxe.Timer(50);
        //Keep calling zoomLeft every 50 ms
            timer.run = zoomLeft;
            zoomLeft();
                                        //And call it once just for instant
response
        }

        la.onMouseUp = function()
            //When they stop holding the mouse
        {
            timer.stop();
                                    //stop scrolling!
        }

        ra.onPress = function()
            //When someone clicks the right arrow...
        {
            timer = new haxe.Timer(50);
        //Well... you know
            timer.run = zoomRight;
            zoomRight();
        }
        ra.onMouseUp = function()
            //Surprise
        {
            timer.stop();
        }
    }



    //---------------------------------------------------------------------------
    // zoomLeft
    //
    //      Move the graph left (i.e. the graph shows an earlier time frame)
    //---------------------------------------------------------------------------
    static function zoomLeft()
    {

            current = false;
                    //We aren't looking at current data anymore
            endTime -= zoomInterval;
            //Move the right side of the graph back by zoomInterval time
            ra._visible = true;
                    //Show right arrow so they can get back
            if(endTime - timeFrame <= firstTime)
            {
                endTime = firstTime + timeFrame;
    //We reached the begining of the graph stop scrolling
                la._visible = false;
                timer.stop();
            }

            redraw();
            Problem.shiftProblems(firstTime - endTime);

    }



    //---------------------------------------------------------------------------
    // zoomRight
    //
    //      Move the graph right (i.e. the graph shows a later time frame)
    //---------------------------------------------------------------------------
    static function zoomRight()
    {

            endTime += zoomInterval;
            //Looks a lot like zoomLeft, except...
```

```
            la._visible = true;
            if(endTime >= lastTime)
            {
                current = true;
                        //We stop when we reach the end of the graph
                endTime = lastTime;
                ra._visible = false;
                timer.stop();
            }

            redraw();
            Problem.shiftProblems(firstTime - endTime);

    }



    //------------------------------------------------------------------------
    // main
    //
    // Incredible initializing
    //------------------------------------------------------------------------
    static function main()
    {
        if(flash.Lib.current.graphDisplayTime){
            displayTime = flash.Lib.current.graphDisplayTime;
        }
        if(flash.Lib.current.graphHeight){
          height = flash.Lib.current.graphHeight;
        }
        if(flash.Lib.current.graphWidth){
          width = flash.Lib.current.graphWidth;
        }

        graph_width = width;

        zoomInterval = (displayTime * 60 * 1000) / 40;
        minutes = displayTime;
        timeFrame = (minutes * 60 * 1000);
        t_interval = (timeFrame / numMarkers);
        p_interval = ((graph_width-15) / numMarkers);
        ttop = ((graph_width-15) / timeFrame);

        var i : Int;

                                                        //
Create connection to JavaScript
        flash.external.ExternalInterface.addCallback("update", null, update);
```

```
              //flash.external.ExternalInterface.addCallback("drawCredit", null, drawCredit);
  //var js = haxe.remoting.Connection.jsConnect();
              var newdate = Date.now();
              endTime = newdate.getTime();
       //Initialize a bunch of stuff
              lastTime = endTime;
              prevtime = lastTime;
              firstTime = endTime;
              labels = new Array();
              labelbg = new Array();
              problems = new Array();
              var mc : flash.MovieClip;
              mc = markerlayer.createEmptyMovieClip("labels", 10);
              for(i in 0...(numMarkers+1))
              {
                     var labelname : String = "label" + Std.string(i);
                     labels[i] = mc.createTextField(labelname, (i+(numMarkers+1)), 20, (height-20),
60, 20);
                     labelbg[i] = mc.createTextField(labelname, (i), 20, (height-21), 100, 20);
                     labels[i].selectable = false;
                     labelbg[i].selectable = false;
                     labels[i].text = "0";
                     labels[i].textColor = 0xFF5555;
                     labelbg[i].textColor = 0xFFFFFF;
                     labels[i].setNewTextFormat(new flash.TextFormat("verdana", 11, null, true));
                     labelbg[i].setNewTextFormat(new flash.TextFormat("verdana", 12, null, true));
              }

              est_pt = Math.ceil((height/2) * 0.75);
              //creditlabel = creditlayer.createTextField("creditlabel", 2, graph_width, 0, (width -
graph_width), height);
  //creditlabel.selectable = false;
  //creditlabel.text = "0";
  //creditlabel.textColor = 0xFF5555;
  //creditlabel.setNewTextFormat(new flash.TextFormat("verdana", est_pt, null, true));

              var temp = new Problem(problemId);
              problems.unshift(temp);
              problems[0].setCorrect(true);
              drawMinutemarkers();
              //drawBackground(100);
              drawArrows();
              //draw background for partial credit window
              //instead of doing this, probably should find some way of seperating which
              //movieclips are used for graph or used for partial credit. that way I don't
              //have to do all sorts of crazy stuff to get the graph to not display over
              //the partial credit area (or could just put a high layer that obscures it.
              //var mc : flash.MovieClip = creditlayer;
```

```
        //mc.beginFill(0xFFEE9B, 100);
        //mc.moveTo(graph_width,0);
        //mc.lineTo(graph_width, height);
        //mc.lineTo(width, height);
        //mc.lineTo(width, 0);
        //mc.lineTo(graph_width, 0);
        //mc.endFill();
        //drawCredit(100);
 }
}



//---------------------------------------------------------------------------
// Problem
//
//  Stores all the information regarding a single Assistment including a list of
//  Point objects.
//---------------------------------------------------------------------------
class Problem
{
    static var prb_bg_id : Int = 4;
    static var prb_pointbase_id : Int = 100;

    var id : Int;
                            //Numeric identity of the problem
    var begintime : Float;
            //Time when the problem began
    var endtime : Float;
                //Time when the problem ended
    var points : Array<Point>;
        //Stores a list of points
    var times : Array<Float>;
            //Stores a list of respective point times (when they were created)


    //Should I move this into the Point class. Since I display by problem I could leave movement to
the point
    var correct : Int;
                //Whether or not the Problem is correct, 0=unknown, 1=correct,
2=wrong
    var bg : flash.MovieClip;
        //MovieClip for drawing colored background on problem
    var complete : Bool;
                //Whether or not the problem is done (for drawing problem division)
    var division : flash.MovieClip;
    //MovieClip for drawing the end of problem border
    var lines : flash.MovieClip;
    //MovieClip for drawing lines connecting points
    var effort : Float;
                        //Number to keep track of student effort on this problem (0-
100);

    //--------------------------------------------------------------------------
    // new
    //--------------------------------------------------------------------------
    public function new(id : Int)
    {
        this.id = id;

        var newdate = Date.now();
        this.begintime = newdate.getTime();
        this.points = new Array();
        this.times = new Array();
        this.correct = 0;
        this.bg = EffortBar.pointlayer.createEmptyMovieClip("problem_background", prb_bg_id
+ id);
        this.complete = false;
        this.division = null;
        this.effort = 100;
    }

    //--------------------------------------------------------------------------
    // addPoint
    //      input:  data    Int
    //                          type    String
    //
    //      Creates a new point in the problem at y-pos (data) of type (type)
    //--------------------------------------------------------------------------
    public function addPoint(data : Int, hover: String, type : String)
    {
        var newdate = Date.now();
        var newtime = newdate.getTime();
        var offset = EffortBar.endTime - EffortBar.firstTime;
        var p = new Point(prb_pointbase_id, data, hover, type);
        var scaled_height : Float = (EffortBar.height - 15)/EffortBar.height;
        p.setPosition(offset * EffortBar.ttop, (data * scaled_height));
        this.points.push(p);
        this.times.push(newdate.getTime());
        prb_pointbase_id++;
        drawLines();
    }

    //--------------------------------------------------------------------------
    // findPoint
//        input:   xpos     Float
//                                                   ypos     Float
//
//        Calls isin for each point in this problem.  If any return true then it
//        activates the hoverbox for the associated point.
//-------------------------------------------------------------------------
public function findPoint(xpos : Float, ypos : Float)
{
        var i : Int;
        var mx = xpos - EffortBar.pointlayer._x;
        var my = ypos;
        for(i in 0...this.points.length)
        {
                if(this.points[i].isin(mx, my))
                {
                        activateHover(i);
                }
        }
}

//-------------------------------------------------------------------------
// clearHover
//
//        Hides the hoverboxes for all points.
//-------------------------------------------------------------------------
public function clearHover()
{
        var i : Int;
        for(i in 0...this.points.length)
        {
                this.points[i].hideHover();
        }
}

//-------------------------------------------------------------------------
// activateHover
//        input:   id        Int
//
//        Hides all hoverboxes then activates the one for point = id
//-------------------------------------------------------------------------
public function activateHover(id : Int)
{
        clearHover();
        this.points[id].showHover();
}

//-------------------------------------------------------------------------
// close
```

```
//
//          Called when the current problem is finished and the division bar and
//          colored background can be drawn.
//------------------------------------------------------------------------
public function close()
{
            var newdate = Date.now();
            this.endtime = newdate.getTime();
            this.complete = true;
            drawDivision();
}

//------------------------------------------------------------------------
// modifyEffort
//          input:   change  Float
//
//          updates and makes sure effort is within 0-100 then triggers an update of
//          the effort background.
//------------------------------------------------------------------------
public function modifyEffort(change : Float)
{
            this.effort += change;
            if(this.effort > 100)
            {
                        this.effort = 100;
            }
            if(this.effort < 0)
            {
                        this.effort = 0;
            }

            EffortBar.updateBackground();
}

//------------------------------------------------------------------------
// getEffort
//
//          Returns the effort.
//------------------------------------------------------------------------
public function getEffort()
{
            return this.effort;
}

//------------------------------------------------------------------------
// drawDivision
//
//          Attaches the Division image.
```

```
    //-----------------------------------------------------------------------------
    function drawDivision()
    {
        if(division == null)
        {
            division = EffortBar.pointlayer.attachMovie("bound", "bound",
prb_pointbase_id);

            prb_pointbase_id++;
        }

        division._x = (EffortBar.endTime - EffortBar.firstTime) * EffortBar.ttop;
        division._height = EffortBar.height;
    }

    //-----------------------------------------------------------------------------
    // drawLines
    //
    //      Draws a line connecting points contained within this problem.
    //-----------------------------------------------------------------------------
    function drawLines()
    {
        var i : Int = this.points.length-1;

        //var offset : Float = this.times[0] - EffortBar.firstTime;
        var offset : Float = EffortBar.endTime - EffortBar.firstTime;
        var scaled_height : Float = (EffortBar.height - 15)/EffortBar.height;
        EffortBar.linelayer.lineStyle(2.0, 0x000000, 100.0);

        if(i == 0)
        {
            EffortBar.linelayer.moveTo((offset * EffortBar.ttop) + 7.5, (this.points[i].data *
scaled_height) + 7.5);
        }
        else
        {
            //offset = this.times[i] - begintime;
            offset = EffortBar.endTime - EffortBar.firstTime;
            EffortBar.linelayer.lineTo((offset * EffortBar.ttop) + 7.5, (this.points[i].data *
scaled_height) + 7.5);
        }

        //var pos = 585 - ((EffortBar.endTime - EffortBar.firstTime) * EffortBar.ttop);
        //EffortBar.linelayer._x = pos;
    }

    //-----------------------------------------------------------------------------
    // shiftProblems
//          input:    pos       Float
//
//          Shifts the layers when an update happens so that it looks like the graph
//          is moving.
//--------------------------------------------------------------------------
static public function shiftProblems(pos : Float)
{
            var change = (((pos / EffortBar.timeFrame)+1) * (EffortBar.graph_width-15)); //pixels
            EffortBar.linelayer._x = change;
            Point.shiftPoints(EffortBar.linelayer._x);
}

//--------------------------------------------------------------------------
// setCorrect
//          input:    value     Bool
//
//          Changes the status of the problem to either correct or incorrect.
//          true = correct, false = incorrect
//--------------------------------------------------------------------------
public function setCorrect(value : Bool)
{
            if(value)
            {
                        this.correct = 1;
            }
            else
            {
                        this.correct = 2;
            }
}

//--------------------------------------------------------------------------
// iscorrect
//
//          Returns whether the problem is correct.
//--------------------------------------------------------------------------
public function iscorrect()
{
            if(correct == 1)
            {
                        return true;
            }
            else
            {
                        return false;
            }
}
```

```
	//---------------------------------------------------------------------------
	// drawBackground
	//
	//		Draws the colored box in the background of the problem space indicating
	//		whether or not the problem was correct.
	//---------------------------------------------------------------------------
	public function drawBackground()
	{
		var offset = (EffortBar.endTime - EffortBar.firstTime) * EffortBar.ttop;
		//var left = ((EffortBar.endTime - this.begintime) * EffortBar.ttop);
		//var right = ((EffortBar.endTime - this.endtime) * EffortBar.ttop);
		var right = offset + 7.5;
		var left = offset - ((this.endtime - this.begintime) * EffortBar.ttop) + 7.5;

		if(this.correct == 1)
		{
			this.bg.beginFill(0x00FF00, 5);
		}
		else
		{
			this.bg.beginFill(0xFF0000, 5);
		}

		this.bg.moveTo(left,0);
		this.bg.lineTo(left, EffortBar.height);
		this.bg.lineTo(right, EffortBar.height);
		this.bg.lineTo(right, 0);
		this.bg.lineTo(left, 0);
		this.bg.endFill();
	}
}



//---------------------------------------------------------------------------
// Point class stores information about a point object.
//---------------------------------------------------------------------------
class Point
{
	var id : Int;
								//Numeric identity of the point
	public var data : Int;
			//Relative graph position (0-100)


	//NOTE: I may want to change data to something less... altered (presevere some original data?)
	var mc : flash.MovieClip;
			//MovieClip that stores the image of the point
```

```
    //var ht : flash.MovieClip;
            //MovieClip that stores the hover box
    var ht : flash.TextField;


    //NOTE: I may change this just to a text box, since I'm not rendering anything fancy.
    //dunno if I need this
    var type : String;

    //--------------------------------------------------------------------------
    // new
    //--------------------------------------------------------------------------
    public function new(id : Int, data: Int, hover: String, type : String)
    {
        //Initialize
        this.id = id;
        this.data = data;
        this.type = type;

        //Create point image
        this.mc = EffortBar.pointlayer.attachMovie(type, "point", id);

        //Create hoverbox
        //this.ht = EffortBar.hoverlayer.createEmptyMovieClip("hoverbox", id+500);
        this.ht = EffortBar.hoverlayer.createTextField("hoverbox", id+500, 20, 80, 50, 18);

        //Set point properties
        this.mc._x = 0;
        this.mc._y = 0;

        //Set hoverbox properties
        this.ht._visible = false;
        this.ht.autoSize = true;
        this.ht.selectable = false;
        this.ht.multiline = true;
        this.ht.textColor = 0xFFFFFF;
        this.ht.background = true;
        this.ht.backgroundColor = 0x888888;
        this.ht.setNewTextFormat(new flash.TextFormat("arial", 11, null, null, null, null, null,
null, "center"));
        this.ht.html = true;
        //Fix this to use a var for the date obj
        //this.ht.htmlText = Date.fromTime(EffortBar.lastTime).getHours() + ":" +
Date.fromTime(EffortBar.lastTime).getMinutes() + ":" +
Date.fromTime(EffortBar.lastTime).getSeconds();
        this.ht.htmlText = hover;
        //trace(this.ht._width);
        //trace(this.ht.textWidth);
```

```
		//Set up events for point image
		this.mc.onRollOver = function()
		{
			EffortBar.rollOverEvent();
		}

		this.mc.onRollOut = function()
		{
			EffortBar.rollOutEvent();
		}
	}

	//--------------------------------------------------------------------------
	// About to be made useless... I don't think so
	// Pretty sure this is still very important. (check that shift layer thingy)
	//--------------------------------------------------------------------------
	static public function shiftPoints(pos)
	{
		EffortBar.pointlayer._x = pos;
	}

	//--------------------------------------------------------------------------
	// showHover
	//
	//	Update the position and set the visibility of the hoverbox to true.
	//--------------------------------------------------------------------------
	public function showHover()
	{
		this.ht._x = this.mc._x + 7.5 + EffortBar.pointlayer._x;
		this.ht._y = this.mc._y - this.ht._height + 7.5;
		var hoverright = this.ht._x + this.ht._width;
		if(hoverright > EffortBar.graph_width)
		{
			this.ht._x = this.ht._x - this.ht._width;
		}
		if(this.ht._y < 0)
		{
			this.ht._y += this.ht._height;
		}
		this.ht._visible = true;
	}

	//--------------------------------------------------------------------------
	// hideHover
	//
	//	Set visibility of hoverbox to false.
	//--------------------------------------------------------------------------
```

```
public function hideHover()
{
        this.ht._visible = false;
}

//------------------------------------------------------------------------
// isin
//      input:  mousex  Float
//                                      mosuey  Float
//
//      Checks to see if the mouse position is within the bounds of the point image.
//------------------------------------------------------------------------
public function isin(mousex : Float, mousey : Float)
{
        //Check to see if the mouse X position is within the image bounds
        if(this.mc._x <= mousex && this.mc._x >= (mousex-16))
        {
                //Check to see if the mouse Y position is within the image bounds
                if(this.mc._y <= mousey && this.mc._y >= (mousey-16))
                {
                        return true;
                }
        }

        //Else
        return false;
}

//------------------------------------------------------------------------
// setHoverText
//
//      Changes the text of the hoverbox.
//------------------------------------------------------------------------
public function setHoverText()
{
        trace("set hover text doesn't really do anything yet...");
}

//------------------------------------------------------------------------
// setPosition
//      input:  x       Float
//                                      y       Float
//
//      changes the position of the point image.
//------------------------------------------------------------------------
public function setPosition(x : Float, y : Float)
{
        this.mc._x = x;
```

```
			this.mc._y = y;
		}
	}
```

Appendix F. Effort Bar Browser Compatibility [JavaScript]

```
var major_safe_version = 8;
var revision_safe_version = 42;


var has_safe_version = false;
var version_major = 0;
var version_revision = 0;

var graph_code_str = '<object\
	classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"\
	codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version
=8,0,42,0"\
	width="600"\
	height="50"\
	id="effortGraph">\
	\
	<param name="movie" value="/swf/effortbar.swf"/>\
	<param name="FlashVars" value="graphDisplayTime=5"/>\
	<param name="allowScriptAccess" value="always" />\
	<param name="quality" value="high" />\
	<param name="salign" value="lt" />\
	<param name="scale" value="noscale" />\
	<param name="menu" value="false" />\
	<param name="bgcolor" value="#ffffff"/>\
	\
	<embed\
		src="/swf/effortbar.swf"\
		FlashVars="graphDisplayTime=5"\
		quality="high"\
		salign="lt"\
		width="600"\
		height="50"\
		align="middle"\
		scale="noscale"\
		menu="false"\
		bgcolor="#ffffff"\
		name="effortGraph"\
		swLiveConnect="true"\
		allowScriptAccess="always"\
		type="application/x-shockwave-flash"\
		pluginspage="http://www.adobe.com/go/getflashplayer"\
	/>\
	\
</object>';

// IE
```

```
if (window.ActiveXObject){
    try {
        flashObj = new ActiveXObject("ShockwaveFlash.ShockwaveFlash");
        version = flashObj.GetVariable("$version");
        version = version.split(" ")[1];
        version = version.split(",");
        version_major = parseInt(version[0]);
        version_revision = parseInt(version[2]);
        if (version_major == major_safe_version && version_revision >= revision_safe_version)
{
            has_safe_version = true;
        }
        if (version_major > major_safe_version) {
            has_safe_version = true;
        }
    } catch (e) {
    }
// Firefox
} else {
    try {
        version = navigator.plugins["Shockwave Flash"].description;
        version = version.split(" ");
        version_major = parseInt(version[2].split(".")[0]);
        version_revision = version[3];
        if (version_revision == ' ') {
            version_revision = version[4];
        }
        version_revision = version_revision.split('');
        version_revision.shift();
        version_revision = version_revision.join('');
        if (version_major == major_safe_version && version_revision >= revision_safe_version)
{
                has_safe_version = true;
            }
        if (version_major > major_safe_version) {
            has_safe_version = true;
        }
    } catch (e) {
    }
}

function graphDisplayCode() {
    if (has_safe_version) {
        return graph_code_str;
    } else {
        return "";
    }
}
```

Appendix G. Random Survey Prototype [JavaScript]

```
// new Survey([number], [SurveyCategory])
function Survey(likelyhoods, categories) {
    var likelyhoods = likelyhoods;
    var categories = categories;

    this.display_test = function() {
        for(var i=0;i<categories.length;i++) {
            alert(categories[i].name() + ' [ ' + likelyhoods[i] + ' ]');
            categories[i].display_test();
        }
    };

    this.pickQuestion = function() {
        //rands and such
        //XXX
        //alert((Math.random() * categories.length)-0.000001);
        return categories[Math.floor((Math.random() * categories.length)-
0.000001)].randQuestion();
    };
}

// new SurveyCategory(string, [SurveyQuestion])
function SurveyCategory(name, questions) {
    var name = name;
    var questions = questions;

    this.name = function() {
        return name;
    };

    this.display_test = function() {
        for(var i=0; i<questions.length; i++) {
            questions[i].display_test();
        }
    };

    this.randQuestion = function() {
        return questions[Math.floor((Math.random() * questions.length)-0.000001)];
    };
}

// new SurveyQuestion(string, string, [string])
function SurveyQuestion(prompt, type, answers) {
    var prompt = prompt;
    var type = type;
    var answers = answers;
    this.display_test = function() {
        //build answers
        var answer_text = '[';
        for(var i=0;i<answers.length-1;i++) {
            answer_text = answer_text + '"' + answers[i] + '"' + ', ';
        }
        answer_text = answer_text + '"' + answers[answers.length-1] + '"';
        answer_text = answer_text + ']';
        //show
        alert('['+type+'] '+prompt+'\n'+answer_text);
    };

    this.prompt = function() {
        return prompt;
    };

    this.type = function() {
        return type;
    };

    this.answers = function() {
        return answers;
    };
}

// Affect - General
afg = new SurveyQuestion("Which of these best describes your emotion, right before this box popped up?", "choice", ["bored", "confused", "delighted", "focused", "frustrated", "neutral", "other"]);

// Affect - Frustration
aff1 = new SurveyQuestion("Are Ye Mad!?", "scale", ["definitely no", " ", "neutral", " ", "definitely yes"]);
aff_list = [aff1];

// Attitude - Tutor
att1 = new SurveyQuestion("Do you like us?", "choice", ["Yes", "Maybe", "No"]);
att_list = [att1];

freedom_survey = new Survey([0.5, 0.5],
                                                [

    new SurveyCategory("Affect - Frustration", aff_list),

    new SurveyCategory("Attitute - Tutor", att_list)
                                                ]);
//auto-indent is annoying

var colors = ['#FF2200', '#FF8800', 'yellow', '#88FF00', '#22FF00'];
```

```
function makeAnswerInput(question) {
    var type = question.type();
    var gui = '';
    var answers = question.answers();

    if (type == "scale") {
        gui = '<div id="survey_scale_answer">Select one:<table><tr>';
        for (var i=0; i<answers.length; i++) {
            gui = gui + '<td style="width:100px; height:40px;text-align:center;background-
color:' + colors[i] + ';border:1px solid black;font-size:10pt;" onClick="alert(this);">' + answers[i] + '</td>';
        }
        gui = gui + '</tr></table></div>';
    } else if (type == "choice") {
        gui = '<div id="survey_choice_answer">';
        for (var i=0; i<answers.length; i++) {
            gui = gui + '<label><input type="radio" name="surveyQuestion"/>' + answers[i]
+ '</label><br/>';
        }
        gui = gui + '</div>';
    } else {
        gui = '<div id="survey_choice_answer"><span style="width:100;height:40;background-
color:yellow;">other</span></div>';
    }
    return gui;
}
```