Iterative Design and Implementation of an Educational Platform for Shared Computing Contexts

by Sunil Garg

A senior thesis submitted in partial fulfillment of the requirements for the degree of Bachelor of Science
With College Honors
Computer Science & Engineering
University of Washington
March 2010

Presentation of work given on: ________________________

Thesis and presentation approved by: ________________________

Date: ________________________
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>3</td>
</tr>
<tr>
<td>INTRODUCTION</td>
<td>3</td>
</tr>
<tr>
<td>RELATED WORK</td>
<td>4</td>
</tr>
<tr>
<td>ITERATION ONE: MULTIMATH</td>
<td>5</td>
</tr>
<tr>
<td>PRELIMINARY OBSERVATIONS</td>
<td>6</td>
</tr>
<tr>
<td>Trials</td>
<td>6</td>
</tr>
<tr>
<td>Observations</td>
<td>8</td>
</tr>
<tr>
<td>ITERATION TWO: MULTILEARN</td>
<td>9</td>
</tr>
<tr>
<td>TECHNICAL IMPLEMENTATION</td>
<td>11</td>
</tr>
<tr>
<td>Application Architecture</td>
<td>11</td>
</tr>
<tr>
<td>Keypad Implementation</td>
<td>12</td>
</tr>
<tr>
<td>Logging Database</td>
<td>13</td>
</tr>
<tr>
<td>Data Storage</td>
<td>14</td>
</tr>
<tr>
<td>FUTURE WORK</td>
<td>15</td>
</tr>
<tr>
<td>Deployment and Testing</td>
<td>15</td>
</tr>
<tr>
<td>Educational Value</td>
<td>15</td>
</tr>
<tr>
<td>Localization</td>
<td>16</td>
</tr>
<tr>
<td>Networking and Synchronization</td>
<td>17</td>
</tr>
<tr>
<td>Input Methods</td>
<td>17</td>
</tr>
<tr>
<td>An Edition for the Developed World</td>
<td>18</td>
</tr>
<tr>
<td>CONCLUSION</td>
<td>18</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>19</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>20</td>
</tr>
</tbody>
</table>
ABSTRACT

As the need for access to technology in developing regions increases rapidly, the supply of personal computers in these areas fails to meet the demand. In the context of education and the presence of computers in under-funded schools, the computer-to-child ratio limits equal access to educational material and deprives marginalized children of valuable digital literacy skills. In this thesis, I present MultiLearn, which expands on prior shared computing research to demonstrate that ten-key numeric keypads are usable as primary input devices and provide the benefits of increased engagement and collaborative learning while offering an inexpensive and versatile educational tool for shared computing scenarios. Moreover, MultiLearn's initial trials show that such an educational tool can be used to create a platform with rich features including teacher-created curriculum and effortless student assessment.

INTRODUCTION

For most disadvantaged youth in the developing world, the first and often only source of access to technology is through institutions that offer shared computing facilities. The presence of classroom computers in developing countries is on the rise, even in some of the poorest parts of the world. Government and community funded urban schools in developing nations are included in this technological trend, but only to a limited degree.

Often in these schools there are only one or two computers for the students and teachers to share. These computers are primarily used for educational purposes in addition to some gaming. The current model for educational software typically consists of a narrative or short audio story followed by a series of questions. Prior observations in these schools have shown that up to nine or ten children will share a single computer to complete this type of coursework, inevitably causing some students to be left out, a dominant child taking complete control, and many children never interacting with the software or hardware at all [5]. Research has shown that, despite clear evidence of computer sharing being the dominant means of access among disadvantaged children, there is almost no hardware or software designed explicitly to accommodate such usage scenarios [5].

MultiLearn uses multiple numeric keypads as input devices for a basic learning games and divides a single computer screen into sections corresponding to each student. This setup — individual USB
keypads and corresponding split screen display — provides equal access to the technology, allows students to access personalized and adaptively calibrated content, and lets students participate at their own skill level while maintaining a sense of competition between the other students.

Additionally, MultiLearn allows classroom teachers to independently create content and track individual student performance, thus allowing for locally relevant content and a complete feedback loop which enables them to use results to inform future lessons.

**RELATED WORK**

Research with on-screen keyboards has demonstrated and explored the possibility of text entry using a mouse [4, 7]. However, by using a physical keypad, as in MultiLearn, we save valuable screen real estate and hypothesize that a physical keypad will allow for faster and more accurate input.

As a low-cost input device — roughly four dollars apiece — the USB keypads can be deployed cheaply and easily on any computer with USB input capabilities. In addition, the keypads provide a versatile platform for various input types, as they can support numeric input as well as directional input, multiple choice questions, and key mapping for text entry.

The concept of single display groupware (SDG) is not a new idea, and researchers have previously explored the potential benefits of collaboration through multiple input devices [2]. Our research follows the SDG model and provides an easily extensible foundation for adding more educational content over time.

Prior research on using multiple mice has demonstrated the benefits of providing each student with their own input device [5]. The overall engagement of the students with the material was higher when each child had their own mouse. We observed similar results in preliminary trials, described below, with the keypads. The chaotic talking and physical contact between students decreased significantly when each child had their own device, and the perceived learning was higher in this scenario as well.

There has also been research done using a split screen model [3], where the screen was split into two halves and the children worked in teams of two. The results of this study showed that by giving each child their own input device, the common scenario of one dominant child controlling the mouse or keyboard was reduced and collaboration among the teams increased, and our own work
has explored this trend as well [8]. Research has also shown that children pay almost no attention outside of their section of screen [3], which we have found to also apply when the screen is split into four sections rather than two.

**ITERATION ONE: MULTIMATH**

Initially motivated by the simple question of whether numeric keypads were a viable input devices for shared computing in the educational context, we developed MultiMath, a simple multiplayer arithmetic drill game.

The MultiMath software is designed as a competitive game that tests students on basic arithmetic operations. The students are initially presented with a screen split into four sections colored blue, green, yellow, and pink. There is a progress bar across the top of the screen which indicates each student’s current progress in the race. The multiple progress bars and split screen keep students focused on their work while allowing them to stay involved in the competitive aspect of the game.

The split screen setup also allows each student to work at their own skill level and allows us to employ adaptive questioning in the form of difficulty scaling to simultaneously challenge and encourage students. MultiMath automatically generates the arithmetic problems presented to students guided by some basic rules for difficulty scaling which enable the software to present students with questions of appropriate skill level.

In applying and extending upon forth the past findings from various iterations of multiple input interactions, MultiMath pursues three courses. First, we maintain the racing aspect to ensure that the engagement gains from multiple inputs persist. Second, we use adaptive questioning to create competitiveness and reduce the impacts of lost attention by concurrent players who are progressing at different speeds. Third, we build upon the shared screen model to both demonstrate the usability of multiple windows and exploit the potential for ‘personal screen real estate’ for young children.
PRELIMINARY OBSERVATIONS

In March 2009, we conducted initial field trials on our first iteration of MultiMath. These studies took place at four government-run primary schools in Bangalore, India, which are attended entirely by children from a low-income demographic. The main purpose of the initial trials was to provide feedback on the initial design, gain experience with developing and implementing appropriate and effective testing methodologies, and to start understanding the important physical and social factors in our target deployment environments.

In addition, we wanted to answer some high-level proof-of-concept questions:

1. Can students understand and quickly grasp the numeric keypad as an input device?
2. How is learning and engagement affected by competition, collaboration, or a combination of the two?
3. How usable is the screen when split into four sections?

Trials

We tested MultiMath with 35 groups of four students each, 140 students in total. The groups were comprised mostly of fourth and fifth grade students; however, we ran a few trials with students in second and third grade to determine the appropriate difficulty level of our content. The groups ranged from all girls or all boys to mixed gender groups, which produced varying social interactions that have been documented and discussed in prior research [5]. The students dominantly spoke their local language, Kannada, in addition to some rough English. Our interactions were thus facilitated by two translators who were familiar with the schools and similar work with both software and children. We used two laptop computers to run simultaneous trials. Our testing methodology was fairly fluid and the procedure and questions were adjusted to provide results that would more directly answer our high-level questions.
When the students entered the room in groups of four, they would seat themselves on the bench in front of the laptop and color coded keypads. The children were encouraged to seat themselves because as past findings indicate, patterns of screen and input dominance emerge in the ways the “central position” seating works out within a group [5], and we wanted to verify and record these interactions within our own observations.

The translator would then briefly explain the game. Over the course of our trials we also had the translator explain the use of the “enter” and “backspace” keys because there was some confusion over their function or existence. The children were then allowed to play and interpret the game as they saw fit. Each group of students was exposed to three versions of the software. The first trial with a single group consisted of each child having their own keypad and the screen being divided into four sections, which were color coded to correspond with the keypads. The second trial with the same group involved the children dividing into two teams of two with each team sharing a keypad. This trial combined the competitive aspect of our game between the two teams with collaboration within a single team. In the final round with this group, all four students shared a single keypad.

Upon completing three rounds of the game, we asked the students a series of questions:

- Which round did you like more?
- Did you like racing against each other?
- Were the problems difficult?
- Did you like doing math problems on the computer rather than with paper and pencil?
- What other kinds of questions would you like?
- How often do you use a computer?
- How many people are using the computer with you when you use it?
- If this game was on your school computers, would you use it?
- Would you use it more or less than other games you play at school?
- What would you change about our game?
- Would you like to know the right answer after you answer a question incorrectly?
- Do you want to know more about computers?
- What do you want to be able to do with a computer?

Occasionally, we would ask additional questions to highlight a specific behavior that emerged during the trial or to more fully understand the dynamics within the group (e.g. social factors such
as friend groups or popularity within a specific group.) The children were then free to go, however many of the students instead wanted to play the game again. When all the groups were finished at each school, we conducted a final Q&A with the educators to solicit their feedback and opinions on our game and the feasibility of integrating it into their classrooms.

Observations

By observing the children’s interactions with the keypad, we concluded that the keypads were a viable and suitable input mechanism. The confusion regarding the “enter” and “backspace” keys did not persist after explanation by the facilitators. We had anticipated that the triple-zero key and the numlock key would cause some problems for the students as well, but there was only one instance in which the numlock key was pressed, and despite several instances of the triple-zero key being pressed, students were able to independently correct their mistakes.

In response to our second question, we concluded that despite the opportunities for both competition and collaboration, we could not determine a statistically significant preference for one or the other. We observed students collaborating and remaining uninterested with the competition even when each student had a keypad. We also observed tremendous excitement over the competition and audible gratification when a student won. The varying results of how children viewed and valued competition and collaboration needs further investigation, but as an initial conclusion, we decided that offering flexibility to compete and/or collaborate seems appropriate.

The trials also illustrated that the 4-way screen division and color coded sections provided an intuitive interface for the game and allowed each child to take ownership of the screen. We did discover, however, that the single progress bar along the top did not draw the children’s attention away from the individual screen sections, which may have been a factor in the students’ lack of interest in the competition. Drawing on these observations and similar results from prior research [3], future iterations of our project give each student a progress bar in his or her screen section so that attention can be focused on one screen section, but on both the problem set and the progress bar simultaneously.

Teacher participation was something we had not extensively considered or anticipated before our trials. However, teachers became actively involved in students’ participation in the game, offering them assistance when needed or helpful hints on how to solve a problem. These observations were encouraging not only because teachers were involved and excited during the game, but teachers
audibly noted that they were surprised by what individual students knew or didn’t know and that the ability to watch students gave them insights on what to focus their future lessons on.

In the final Q&A with the teachers, many of them mentioned wanting to be able to see more of how individual students were doing and what kinds of problems they were struggling with. In addition to this feedback, teachers went further to say that they wanted to create and expand the content the students were using to other subjects and harder problems. This feedback in combination with the successful acquisition of useful data from our basic single-game-logging implementation, motivated the creation of our Teacher Dashboard feature in the second iteration of MultiLearn, which gives teachers control of content creation and statistics on students progress and performance.

Our high-level observations support our hypotheses that multiple keypads and a four-section split screen display are a usable and beneficial solution in shared computing environments.

**ITERATION TWO: MULTILEARN**

Motivated by our findings in India, the next iteration of this application held three primary goals: first, to create additional games which allow for questions in more subject areas as the teachers requested; second, to allow teachers to independently create their own problem sets for these games, to allow for highly relevant localized content; and third, to provide feedback and statistics to teachers regarding student performance.

For our second game, we created a spelling game called MultiSpell, which is similar to MultiMath. This game displays nine letters on the screen and a picture, under which is a series of blanks where the name of the object in the picture will be filled in. The students use the keys 1-9 to select the corresponding letter on the screen to fill in the blanks and spell the word.

The choice to build a spelling game was motivated by our interest of whether text-entry is viable using these keypads. Essentially, each letter in the word being spelled is a 9-way multiple-choice question. Because the text entry is not entirely free-form, we can handle each letter in a single

**Figure 3: MultiSpell**
key-press, rather than multiple key presses as required by similar text-input mechanisms such as T9 on cellular phones.

MultiSpell demonstrates the flexibility and ease with which the USB keypads can be expanded to support numerous input types and curriculums, but there is clearly additional scope for future work targeting free-form text entry using numeric keypads.

Beyond spelling, we have also begun to develop a multiple-choice game, which would allow for questions in any subject area that a teacher might want to cover. We envision this to be similar to the spelling game, where there would be a prompt, related image, and a variety of text or image answers to choose from.

In order to allow teachers to create game content, we have created a separate application called the Teacher Dashboard, which offers the facility to create spelling and multiple-choice problem sets. This application offers a simple interface that enables teachers to import images and enter questions or answers as they pertain to any given question. We save these problem sets for future use in a proprietary file format, which can easily be copied between computers.

This second iteration also offers the functionality for students to login to the game and thus log their performance across sessions. Thus, we are able to report statistics regarding individual student performance and difficulty of individual problems in the Teacher Dashboard.

With the addition of these features, MultiLearn offers a complete content and feedback loop, allowing teachers to create problem sets, let students play them, and use classroom statistics to inform future lesson plans, whether in the classroom through traditional teaching methods or with the creation or adaptation of MultiLearn problem sets.

Another iterative change was in response to the students’ lack of attention towards the progress bar across the top of the game screen. Instead of a single horizontal progress indicator, we have created
identical vertical progress bars within each player’s section of the screen, so that they do not have to shift their attention away from the actual gameplay.

**TECHNICAL IMPLEMENTATION**

MultiLearn is built on Microsoft’s .NET framework as a desktop application for the Windows operating system. Windows already supports multiple input devices, but the input data is processed as a single stream by default, and thus all of the mice control the same cursor and all of the keyboard data is displayed as if it came from a single keyboard. In order to obtain and independently process data from each of multiple input devices, we identify events such as key presses or mouse clicks, and correlate them with a specific device ID. We use this information, which is provided by the Windows RawInput API, to pair users’ personal keypads with the input data from that keypad and tailor the corresponding screen section to react appropriately.

**Application Architecture**

It was always evident that we would want to support not only multiple games, but also multiple types of input devices. Supporting the easy addition of both input devices and games independently of each other would enable us to quickly prototype new ideas for testing.

However, given how attached game-type applications are upon specific forms of input, a lot of careful thought had to be put into the abstraction of how different components of the application speak to each other. The application depends heavily on the .NET programming concept of events – essentially a well-defined, more strongly typed way to implement function pointers. It provides an easy way to add and remove events from certain triggers.

The simplest component prompts for the number of players and the type of input desired. The designated player count is then handed off to the appropriate input managing class – these are defined according to a fairly loose interface to standardize the return type. The input manager is

---

**Figure 5: Illustration of application architecture and communication over time**
responsible for setting things up, including the presentation of a user interface – in the case of keypads, this is the sync dialog.

After the necessary work is done, the input manager will construct an array of input proxies. Each of these input proxies is tied to a specific input device, and contains an event which consumers can listen to, which fires whenever an input is made from the device in question. Each input is given to the consumer in the form of an input data object, which merely contains information about the character representation of what the input meant, as well as flags for special input events such as cancel or confirm (e.g. escape or enter, respectively). A future addition to this data structure interface will likely be point-on-screen data, for mouse clicks or simulated mice using arrow keys on keypads.

Next, the login form receives these input proxy objects and attaches each one to the correct portion of the screen by color. This enables for certain input methods to print custom UI on the screen, such as a virtual on-screen keyboard for mouse input. The login form confers with the database to log users in, and then, as it constructs user objects from the database, it attaches the appropriate input proxy to each so that forms can very easily listen for input coming in from a specific user. This eases implementation of games.

Games are given the same UserControl for custom input UI, as well as the user array obtained from login. While there is no formal, strongly-typed imposition of how to implement each individual game, split-screen games are all generally handled in the same way. The game class itself contains scoring logic, and initializes and lays out a UserControl for each user in the game. The controls deal with user input, obtaining and prompting questions, and determining question correctness. Database logging is implemented such that question logging falls upon the control, and session logging falls upon the game. In this way, the separation of concerns is fairly strong and extends to all aspects of operation.

**Keypad Implementation**

While Microsoft offers a MultiPoint SDK for developing applications with multiple mice, the SDK is written for the Windows Performance Framework (WPF) rather than the WinForms UI framework, and thus has a much higher overhead, requiring more powerful computers to run. Though WPF provides many desirable features such as easy text scaling and vector graphics, we ultimately want
to maximize compatibility for computers already deployed in our target environment, and thus use the framework which itself powers the MultiPoint SDK, the Windows RawInput API.

The RawInput API is actually a COM+ interface, making it difficult to interface with in .NET. Two separate builds are required for 32-bit and 64-bit machines, as the data structure sizes and thus the interop code are different. As is standard for COM, the way we receive messages from the RawInput API is by trapping window messages to the window handle of the application.

.NET normally intercepts these messages before they reach the user-coded portion of the application, marks them as handled, and returns only the keystroke that was indicated. However, there is much more data available about these keystrokes. Each one actually also returns with it several values that are largely internal to Windows – the pointer to the handler of the device, the raw header, and the device handle. The last of these is what we are interested in, as it is unique for each device, and is assigned randomly each time the device is connected.

The RawInput API also provides COM hooks for querying for an explicit list of connected devices, as well as getting more information on a single device. In addition, as is shown by the MultiPoint SDK, the API also provides enough information to process mouse events.

One unresolved issue which arose is that we lose track of which user a keypad refers to when it is accidentally disconnected during play and reconnected, due to the abovementioned fact that device handles are assigned randomly upon each connection and thus do not persist across the connection loss. Currently, we handle this by polling for the list of valid keyboard-type devices every second and comparing the count each time, raising a warning message when one is lost. In the future, we should be able to simply redisplay the keypad synchronization dialog and allow resumption of play.

Logging Database

Currently, we use a SQLite database to store the logging data generated by the application for statistics and feedback purposes. The advantage to SQLite is that its libraries are dynamically linked into the consumer application’s own binary, and thus, unlike most database engines, it does not require the installation of a separate server process and establishment of connections to said process. This has several advantages for our applications and target environment, especially simplified installation and troubleshooting, and lower overhead on the system, both when the game is in use and not.
The database has only a couple of core types, users and sessions, and a join table between the two, which stores a lot of metadata about the user within the session. Everything else is stored in custom tables for each game type.

As it currently stands, it is not possible to add new games to the application without modifying the schema of the database. Since games aren’t currently as interchangeable as plugins, particularly from the perspective of the end-user, we believe that this is a reasonable design decision. As it stands, each game has the MultiLearn project as a build dependency, and must be explicitly added to the application’s menu system. Additionally, statistics must be fetched through custom LINQ queries, so binaries must be modified on that end as well. Finally, since each game might need to store different types of data (e.g. a math game must store the arithmetic operator for each question given), and we would like to preserve the speed optimizations made by the database system, it is best to build the schema specifically for each game rather than abstracting fields out.

An important item to note is that the database schema was built with the task of logging in mind. Thus, since data is only ever written to the database, and never modified, there are several places where data is stored in multiple places to ease performance concerns with querying the data in the statistics application later on, since data and referential integrity will never be in question.

**Data Storage**

With the addition of custom problem sets, we needed a method of easily creating and storing this data to disk. To facilitate this, we have written a library which specifies each type of problem as a C# object which can be encapsulated as a collection in a problem set class. These classes are then be saved as XML using .NET’s built-in support for object serialization.

Additionally, each problem set may contain one or more images, such as the images associated with each spelling question, so we package a problem set’s XML file with its associated image files into a single ZIP file, using an open-source library called DotNetZip. These files are easily portable between systems. This scheme is similar to that employed by Microsoft with the file formats introduced with Office 2007.

After these files are created, upon being opened in the future, they are unzipped to a temporary directory, and the XML file is deserialized into a .NET object which can then be directly used by the game or content creation application.
The existing games and Teacher Dashboard have been integrated with this code, and it has proven to be an effective and simple way of storing problem sets. It is also easily extensible to support future games and data types.

**FUTURE WORK**

While MultiMath is now a very complete product in and of itself, since our refocus as a learning platform we have found increasingly many features which we would like to add to the application. Most of these revolve around either packaging and deployment or the content generation aspects of the game.

**Deployment and Testing**

The application is fully capable of functioning on its own at this point, and we have developed an installer for the application. However, it's not yet well-tested, and there are known issues with the application and certain types of hardware, especially 64-bit architectures. Eventually, we will need to build checks into the installer to prevent the application from being loaded on incompatible machines altogether, which would be especially important for large-scale deployments where we are not available to troubleshoot individual machines.

The application also needs to be tested on a more diverse array of hardware, as its current use has been generally limited to machines that the developers personally own. None of the developers have older machines that fit the profile of the computers we typically found in the labs of the schools we visited, and which have seen the same level of software and spyware abuse.

We account for the most common screen resolutions in our application, down to 1152x864, as most of the monitors we encountered while field testing were either flat panel displays capable of 1280x1024 or CRTs running at the same. However, it is a near certainty that the application will eventually be installed on a machine running at 800x600 or worse, and we will need to account for that. In addition, while the application has been used with Windows XP, Windows Vista, and Windows 7, it has not been tested for Windows 98 or ME compatibility.

**Educational Value**

As participation in this project has been generally limited to computer science researchers, we have approached learning aspects of our software from a very cautious standpoint, and not built
anything that would attempt to teach students without the direct input of the teacher. However, there are many features that have been requested by teachers, which would require input from a pedagogically aware adviser in order to design and develop as features for our software.

Chief among these changes is the addition of a tutorial mode. Currently, our game excels only at drilling students' knowledge, not teaching the students new material. Thus, if a student's primary issue is a lack of practice, MultiLearn will be an excellent tool for them. However, our recent studies have shown that most students in our target environment are struggling to understand basic concepts – for these students, playing the game would likely not result in measurable impact with regards to learning. Currently, this problem is addressed by means of our statistics system, which should provide the teacher with the feedback that a particular student is severely struggling, at which point she can step in and provide the guidance necessary to bring the student up to standard. However, it would be useful if our program included features – such as visual depictions of math questions, for instance – which help to gently nudge students in the proper direction and reinforce basic concepts as necessary.

It would also be very desirable for the application to come prepackaged with a large number of suitable problem sets, so that teachers can see the application working and understand what works well in the application before attempting to add their own content to it. Again, we have not done this ourselves given that we do not have the expertise to create pedagogically correct curriculums. Being able to work with someone to develop that content would go a long way towards making our application attractive and deployable.

Recently, we have worked to improve our difficulty scaling algorithm, and recently conducted trials of a model that uses partially observable Markov decision processes to optimally select questions based on each student’s skills, as inferred from their logged history. There is still much room for additional work that could be done here to improve the adaptability and learning outcomes of our software.

**Localization**

While we have limited the amount of text in the interface of both the game and the teacher dashboard, it would be desirable to provide localized language versions of the application. .NET allows for easy creation of localized versions, through XML resource files. We currently have not
implemented an interface to choose and load them into the interface, but once we actually have the translations in the resource files, it will be reasonably straightforward to add the translations in.

Eventually, we would also like to perform tests between various interface languages. It would be useful to formally compare localized, text-free, or limited-English versions of the interface, as limiting the amount of localization work required can greatly reduce the amount of time and effort required for new deployments.

**Networking and Synchronization**

Currently, because we cannot depend on the machines in schools being networked together, we are not able to synchronize logging, login, or problem set data between machines very easily. One solution that is being considered is to supply a small-capacity USB flash drive with a sentinel marker that we can detect for when USB device events occur. This way, when our flash drive is inserted, we can automatically copy all of our data onto the drive. There is a lot of algorithmic work that would have to be done for ad hoc synchronization of files and logs, but ideally the teacher would just have to insert the drive into each machine in order to synchronize state across all of them.

**Input Methods**

There is more that can be done with different input devices, and even with the keypads that we currently use. A common suggestion we have had is to use T9- or ABC-style text input as is found on mobile phones. This would not be difficult to implement, using the custom input UI extension onto input forms. However, we are not sure how intuitive the input method will be, especially for children who may not have had extensive contact with mobile phones, and because the physical keys themselves won’t have the proper letters printed on them. We would like to run formal tests to compare these input methods, extending upon similar work done with multiple mice [7].

We have also created a test application for mouse input that will enable us to directly and formally compare the differences in input speed between mice and keypads, particularly for numeric or letter input. We plan to build an analogous set of games within the MultiLearn interface, and then run formal tests with children.
An Edition for the Developed World

A lot of interest has been expressed in deploying the application in the United States. Removing the constraints we hold ourselves to in terms of hardware and software requirements also greatly improves the scope of what we will be capable of doing with the application.

An obvious differentiator is access to the Internet. With this capability, we are able to move all of our data storage to the cloud. This would mean that we could use a more performant database server running on a dedicated machine, log students in across multiple machines without worrying about synchronization of data, save and load question sets in the cloud and enable public sharing and social discovery of them, and build a robust, web-based statistics viewer rather than relying upon Microsoft’s charting libraries and custom WinForms.

Since we can count on machines being networked to each other as well as the Internet, we can also play larger-scale games through networked play, for example allowing for multiple teams of 4 students, with each team on a single computer. A group of undergraduate HCI students at UW is currently investigating what a “developed world edition” of MultiLearn might look like in order to best address the differing environment.

In addition, without extreme cost limitations, we can look into more advanced or interesting input methods, such as Wii remotes. We can even consider building a WiiWare version of the game so that students can play the learning games at home, and the results are logged to the cloud. Along these lines part of a graduate project course at UW, one student developed a MultiLearn prototype for Xbox.

CONCLUSION

The preliminary field trials, although small in scope, demonstrated that MultiLearn is a usable and useful educational tool for disadvantaged students needing to share computers due to resource constraints. By using low-cost hardware and modularized software, MultiLearn provides the basis for an extensible educational software platform that can easily adapt to existing curriculum in diverse environments.

With another iteration of development, we have shown that MultiLearn can extend beyond mathematics to allow for educational games in virtually any subject area, while allowing teachers the control to create their own curriculum for use in the game.
Finally, teachers are provided with feedback about student performance in a manner that they have not currently achieved with traditional teaching methods. This directly allows for the improvement of classroom teaching by enabling responsive adaptation in lesson plans and continued creation of MultiLearn curriculum.

With continued research and more extensive field tests, there is potential to deploy MultiLearn systems in developing world schools as a high quality educational tool capable of promoting social integration and personal advancement through increased exposure to academic material and technology.

ACKNOWLEDGEMENTS

This work would not have been possible without the assistance and support of Richard Anderson, Joyojeet Pal, Clint Tseng, Heather Underwood, Charlotte Robinson, Emma Brunskill, Leah Findlater, the UW Department of Computer Science and Engineering, the UW Information School, the UW Undergraduate Research Program, Microsoft Research India, the Azim Premji Foundation, and the National Science Foundation.
REFERENCES


