Agents Help Students in ProgrammingLand

Curt Hill
Department of Mathematics
Valley City State University
Valley City, ND, USA
701 845-7103
Curt.Hill@vcsu.edu

Vijayakumar Shanmugasundaram
Concordia College
Moorhead, MN, USA
218 299-3343
shanmuga@cord.edu

Martina Miteva
North Dakota State University
Fargo, ND, USA
701 388 - 8544
Martina.Miteva@ndsu.edu

ABSTRACT

ProgrammingLand is an online system for delivering content to introductory computer science courses as a substitute for a conventional textbook. Because the system has a large number of exhibits, sometimes students were not finding the material needed. The system was recently enhanced with several agents to direct students to pertinent locations. This paper discusses the capabilities and techniques of these agents.

Preliminary data from the use of ProgrammingLand in two different introductory programming classes is discussed. This data suggests that the agents are successful in aiding students, especially those who are struggling.

Categories and Subject Descriptors

1.1 K.3.2 [Computers and Education]: Computer and Information Science Education – *computer science education.*

General Terms

Design, Experimentation.

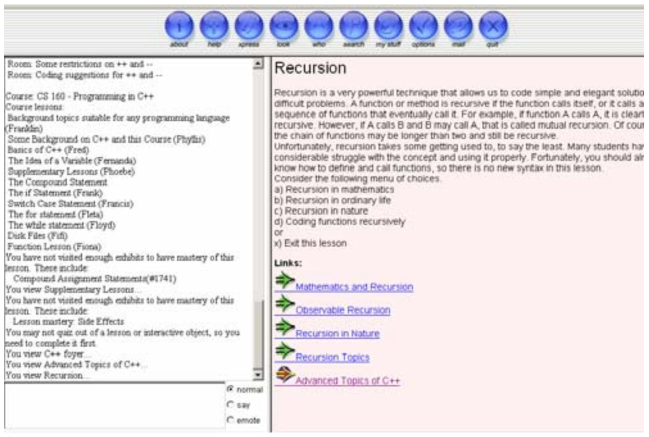
Keywords

CS educational research, online instruction, blended education, distance education.

2. INTRODUCTION

ProgrammingLand [3] is an online instructional system for introductory computer science courses. It contains content material and thus substitutes for a textbook. The system provides testing, communication and administrative features [2] so it also substitutes for a learning management system. The content is organized in terms of lessons, which allow students to process at their own rate and in the order they prefer, making it learner-centered. Lessons are hierarchical and may be contained one within another. The completion of certain lessons prompts an agent to deliver an assignment to the student. A lesson may require a variety of experiences in order to be completed. These may include the simple browsing of textbook-like material, interaction with various types of exercises within the system, and completion of prerequisite or subordinate lessons.

ProgrammingLand is based upon MOO software [1]; thus it is organized into rooms (also known as exhibits) and exits that connect the exhibits. The paradigm is that of a museum where students browse through the exhibits, reading the content material and interacting with educational objects. The original core of the museum and the web browser client is distributed by enCore[4]. A display of the client is shown in Figure 1.



```
Room: Some restrictions on ++ and --
Room: Coding suggestions for ++ and --

Course: CS 160 - Programming in C++
Course lessons:
 Background topics suitable for any programming language
(Franklin)
 Some Background on C++ and this Course (Phyllis)
 Basics of C++ (Fred)
 The Idea of a Variable (Fernanda)
 Supplementary Lessons (Phoebe)
 The Compound Statement
 The if Statement (Frank)
 Switch Case Statement (Francis)
 The for statement (Fleta)
 The while statement (Floyd)
 Disk Files (Fifi)
 Function Lesson (Fiona)
You have not visited enough exhibits to have mastery of this
lesson. These include:
  Compound Assignment Statements(#1741)
You view Supplementary Lessons...
You have not visited enough exhibits to have mastery of this
lesson. These include:
  Lesson mastery: Side Effects
You may not quit out of a lesson or interactive object, so you
need to complete it first.
You view C++ foyer...
You view Advanced Topics of C++...
You view Recursion...
```

Recursion

Recursion is a very powerful technique that allows us to code simple and elegant solutio
difficult problems. A function or method is recursive if the function calls itself, or it calls a
sequence of functions that eventually call it. For example, if function A calls A, it is clear
recursive. However, if A calls B and B may call A, that is called mutual recursion. Of cou
the chain of functions may be longer than two and still be recursive.
Unfortunately, recursion takes some getting used to, to say the least. Many students ha
considerable struggle with the concept and using it properly. Fortunately, you should al
know how to define and call functions, so there is no new syntax in this lesson.
Consider the following menu of choices.
a) Recursion in mathematics
b) Recursion in ordinary life
c) Recursion in nature
d) Coding functions recursively
or
x) Exit this lesson

Links:
Mathematics and Recursion
Observable Recursion
Recursion in Nature
Recursion Topics
Advanced Topics of C++

Figure 1. A display of an exhibit

Much of the original motivation for ProgrammingLand was online and distance education. The student would use the MOO like a textbook, but it would give assignments and ease the administrative load of the distance instructor. It has been used in this mode, but everything that ProgrammingLand provides for distance education, it may also contribute to the normal class taught in a classroom. Thus there is the blended approach of classroom instruction with a significant online support. The average college student does better with both than with only one of the two.

Recent development in ProgrammingLand has progressed on three fronts: the increase of new content material so that more classes may use the system, the implementation of features normally found in a Learning Management System and the implementation of agents that make students more effective in their use of ProgrammingLand. This paper deals with the latter.

The ProgrammingLand MOOseum is one of a series of projects by the World Wide Web Instructonal Committee[5,6].

3. PREVIOUS AGENTS

The oldest agents that existed were of the simplest form. Each time a student entered a lesson room, the student's accomplishments were checked. If the student had completed the requirements and this was a lesson that had an external assignment, then an agent, known as a roving goalie, was dispatched to visit the student. The agent would enter the room, speak to the student, post the assignment on the student's object for future reference and leave. Although the agent was portrayed as a person, the student did not interact with the roving goalie in a more meaningful way that one does with an answering machine.

The types of assignments that a roving goalie gives to a student are typically programming assignments that need to be done outside of the MOO. In an introductory programming class, that is the most frequent type of assignment. Any assignment that can be described in text, some of which may refer to an outside web page, could be delivered by a roving goalie. There are several that give assignments that require the filling in of a worksheet or other possible assignment, but the bulk are programming-style assignments.

Another important aspect of the roving goalie is its ability to give one of a list of equivalent assignments. The roving goalie merely indexes through a list of assignments. If there are more students than unique assignments, it starts over in the list. This greatly discourages the plagiarism that sometimes occurs. The assignments are not given out in any predicatable order, but rather when the student completes the lesson. The program grader may reference which assignment was received for each student and verify that the student turned in the correct solution and not a friend's correct solution.

The roving goalies, however, were and continue to be important to the usefulness of the system. Most instructors have some concern whether students are reading the textbook or not. This may be alleviated in ProgrammingLand by rewarding those students who complete lessons in the system. Lesson completion may be the only mechanism in a course to receive assignments or the students may receive points towards their grade by completing lessons. There are other means as well to monitor how the student is progressing through the material.

4. INTERACTIVE AGENTS

The newer agents all deal with one of several perceived problems that previous students experienced with the system. There are now three new interactive agents that help students. The first gives quizzes to students that could satisfy a lesson without all of the reading; the second searches out lost students; and the third attempts to assist students who are making no progress on their lesson. Each of these will be described in some detail presently.

5. QUIZ AGENT

The plight of the over-experienced student is the problem that the Quiz agent attempts to solve. The introductory class has quite a variation in the material and the level of incoming students. Some of these students have a strong background but may need this class for the programming language that is taught, rather than some of the issues, such as background information or programming skills. The idea of quiz is to allow a more advanced student to take a quiz instead of plowing through all the requirements (so that boredom does not set in).

When a student leaves a lesson room, the system compares what he or she have done with the requirements of the lesson. If the only thing lacking for completion of the lesson is visiting one or more rooms, then the Quiz agent is summoned to talk to the student. Like any agent, it enters the room and speaks to the student. The student is given the offer to take a quiz to prove mastery. If the student accepts, the agent transports the student to a specially designed quiz room. The room accepts a command to give a five question, multiple choice quiz. If the student answers four or more questions correctly, he or she is given credit for the entire lesson. A student may only take the quiz twice and is no longer offered the quiz by the agent after two failures.

The quiz itself is randomly generated. Each exhibit that the student has not visited has a store of one or more questions on the content material of that room. Each such question has a collection of one or more right answers and four or more wrong answers. The quiz generator randomly reduces the pool of questions to the needed five and randomly orders the answers. It then gives the questions, accepts the answers and scores the quiz. When the student completes the quiz, he or she may exit the quiz room.

6. AGENT INFRASTRUCTURE

ProgrammingLand is a MOO and like all MOOs has an object oriented structure. All rooms, exits and players are objects that may have their own properties, methods and events. (Methods are called verbs in a MOO.) There are several objects that need to be considered before a discussion of the lost and aimless agents themselves.

The normal room in ProgrammingLand is called a lecture room. Every lecture room has some content material that the student should read; the system records that the student visited the room on the student's object. Currently there are approximately 2500 such rooms in the MOO, each of which contains about one to five paragraphs of text. Two important descendents of the lecture room are the monitor room and its derivative, the lesson room.

The lesson room is typically the head and only way into a cluster of related rooms that deal with a single topic, either narrow or broad. A student entering a lesson room is checked for completion of the lesson. This is the mechanism that sends a roving goalie, if the lesson has been completed by that individual. A student exiting a lesson room for a room outside of the lesson triggers a check of completion of all of the requirements except for one or more exhibits. If the student has not failed the quiz more than once, then the quiz agent is summoned by this event. There are approximately one hundred lesson rooms in ProgrammingLand. Some of the wings of the MOO have not yet been organized into lessons, so the average number of lecture rooms managed by a lesson room is less than the 25:1 implied by the previous numbers.

The first time a student enters a lesson room, the room will display the requirements of the lesson. At any subsequent visit, the student may issue a command to display these requirements again. However, these may only be shown in the lesson room. Each display of a lesson's requirements also shows which have been completed.

The monitor room also does event processing. When a student enters a monitor room, it calls two verbs on the agent object, which will be discussed later. The two verbs check if the student is lost or if the student is aimless. Since lesson is a descendent of monitor room, it does the same checks. There are only about ten monitor rooms in ProgrammingLand. The main entryways into the various wings of the MOO are usually monitor rooms. These include the C++ foyer, the Java foyer and the Background Topic foyer.

The agent object is the ancestor of both the lost agent and the aimless agent. It contains a number of common routines of both of these. The standard agent practice of these two is similar. The monitor room checks whether the student is lost or aimless by calling verbs on the respective agents. If the student is both lost and aimless, then only lost is called. Next, the agent to be summoned is activated. This involves queuing the request if the agent is busy. When the agent becomes free, it enters the current room where the student is, which may not be one of the monitor rooms because of time lags. The agent then asks if the student needs help and moves him or her to an appropriate room if an affirmative answer is received. The most important difference between the two agents is how to determine if a student is lost or aimless and; involve the requirements of the course and lesson.

Each student is enrolled in one or more courses, which are instances of a course object. A course object contains a list of lessons that need to be completed. This is not necessarily an exhaustive list but only those that the course specifically names. Each of those lessons may cite others that serve as prerequisites to this lesson or subordinate to this lesson. Each lesson, whether in the course object or not, refers to a requirements object which lists the things that must be completed in order to satisfy that lesson's requirements. Just as two instructors may require different readings from the same textbook, the requirements for one course may differ from another course with regards to the same lesson.

The final part of the infrastructure is the student object. This is the object that records which rooms have been visited, which lessons have been completed, and which interactive objects have been used, as well as to which course or courses the student belongs. The lost and aimless checking mechanisms are both based upon the comparison between the completed activities of the student and required activities of the course and lesson.

7. LOST AGENT

A characteristic of many MOOs is a spatial orientation. Exits are labeled with directions such as North, South, Up and Down. ProgrammingLand does not use this particular paradigm, so its exits are labeled by content that appears in the destination exhibit. The unfortunate result is that sometime students do not know where they are or how to get to where they should be. Sometimes, students would work on or complete lessons that were not relevant to their course. This resulted in very low correlation between exhibits visited and class performance. The linear nature of textbook, along with an index and table of contents prevents this in the use of that media. ProgrammingLand had two mechanisms for preventing this. By convention, each exhibit has a single exit that should take the student closer to the entryway. This exit has an icon that is of a different color, so that a student may always find a way back to a known location. There is a also a set of web pages on the same server called the lesson map []. Each lesson is shown with a description of the shortest way to get to the lesson, the names of contained rooms and subordinate lessons. Despite these two, students were still wandering into areas irrelevant to the course or too advanced for their current status. Some wandering is desirable – ProgrammingLand is intended to be learner centered – but when the student cannot engage in constructive browsing of the content, a problem arises. The lost agent is the active solution to supplement the passive solutions just described.

The most important function of the lost agent is the is_lost function – the method of determining if the lost agent should intervene. This function must use a variety of information to determine if the student would benefit from a visit. It receives the monitor room that has been visited and the student object.

The is_lost function must first determine if the room entered is a safe room. A safe room is any room that a student might go through that should not arouse attention. For example, when a student logs in to the MOO, the starting point is the main entryway. For a C++ student, the normal path is to enter the C++ foyer and then further into the lesson room of choice. A C++ student who that enters the Java foyer is curious or lost, but the C++ foyer is a safe room. The safe room list is attached to the course object and the lost agent obtains access to it from the student object. Therefore, if the student just entered a safe room, the function determines him or her not to be lost.

The course object is also needed to find the lessons and their requirements. The course contains an ordered list of lessons that the students should complete. This set of lessons may be complete before the beginning of the class or the instructor may only add lessons after the class has progressed to a previous lesson. In either case the lesson that the student should be in is the first lesson that is incomplete. The student object contains a list of events, including completed lessons, so the is_lost function compares that list with the list of lessons to determine where the student should be working. If they are in a previous lesson that is not considered a lost situation, but instead the presumption is that they are reviewing material already seen. If the student is in the next lesson, that is just one beyond where he or she should be that is also allowed. However, anywhere beyond that point or in any area not otherwise covered by the course, the student is determined to be lost.

An affirmative return from is_lost prompts the system to dispatch the agent to visit the student. There is only one lost agent and possibly many lost students, so this request may be queued until the agent is available. However, the time required to assist a student is not very lengthy, so there is seldom more than one student waiting processing.

Upon activation, the lost agent moves to the exhibit the student is currently visiting. Since delays are possible, it does not have to be a monitor room. It then mimics a person, mentioning that the student appears to be lost and asking if assistance is desired. The student will then reply to the question. An affirmative response causes the agent to move itself and the student to an exhibit in the lesson that is incomplete. In practice this could be any one of many such exhibits. The lost agent then exits.

One of the goals of the agents is to avoid being annoying. A student may be in an area that is not pertinent because of losing the way or because of curiosity. The system would not be learner-centered if it forced the student back to the lesson that the system thinks is the appropriate one. Therefore, the question on help is asked. If the student declines help at this point, this fact is recorded on the student object. If the agent visits again after the next pass through a monitor room, it would get annoying to decline help again. Therefore, is_lost will not signal the revisiting of a student that has declined help until the student has visited a fixed number of other monitor rooms. This number is currently set at ten, but may be tuned at a future time. Visiting ten monitor rooms implies the visitation of many other lecture rooms. A logoff from the MOO also clears the counter.

8. AIMLESS AGENT

A student, who is not making progress towards completing the requirements of any lesson is considered aimless. This is a more complicated problem than is seen with the lost agent. The lost agent may examine the course and student and from that only determine if the student is lost. The determination of aimlessness requires all of these things, but must also consider the recent actions of the student.

When a student starts working on several lessons simultaneously, each of which may include lessons, keeping track of the requirements and room locations becomes a challenge. The aimless agent records lesson progress for each student and what that student needs to do next. The purpose of the agent is to make sure students are taking steps towards completing the lesson and navigating through the MOO with a purpose in mind. The agent detects potential aimless students, informs them what they need to do to make progress with their lesson and helps students navigate by transporting them to the next room they need to visit, or interactive object that must be executed.

For each student who enters a new lesson, a progress object is created for that lesson and appended to a list of progress objects attached to the student object. Of course, the student is unaware of the things happening under the surface. When this student completes a lesson, the progress object for that lesson is removed from this list. Thus, the list keeps track of all lessons on which the student is currently working. The requirements list of a lesson depends on the course the student is enrolled in, as well as on the instructor of the course. Therefore, a progress object is created only for lessons required for courses in which the student is enrolled.

Progress for a student would be the visiting of a room or the accomplishment of an event. An event would be the completion of a lesson or a complete interaction with any of several objects. A progress object is related to one particular lesson for one particular student and has one room that could be suggested for the next visit or one interactive object to execute. Either of these could be empty, but not both. When the student completes the lesson, the progress object for that lesson and student is discarded.

Every time the student enters a lesson room, the event and room counts are updated. The MOO records every lecture room a student visits as well as any events, so when a student enters a lesson room, the progress object for that lesson is updated to reflect any activity completed since the student's last visit. The aimless agent decides whether a student is making progress in the current lesson by counting how many times counts of unvisited rooms and incomplete events has remained the same.

The annoyance factor is considered for the aimless agent as well. Like the lost agent, when a student declines help, this keeps the aimless agent away for a number of visits. The aimless agent is also more reluctant to visit the student than the lost agent, in another sense. Any motion in an unrelated area brings the verdict of lost, however one trip through an area without progress does not summon the aimless agent. Instead it requires exceeding a threshold of visits to the lesson room before summoning aimless and this threshold is currently set to five. In those cases where aimless and lost could both be activated, lost is summoned and aimless is not.

Aimless also checks students at login by searching the student's progress list for the next needed event or room and comparing them to the ones on his list. If they are the same, the student has not made any progress. A count for the lesson is increased and when it exceeds the threshold, the aimless agent is also activated.

The aimless agent provides advice based on the current location of the student. If the student is considered aimless at login, and is located in the Entryway outside any lesson, he or she is advised to move to the first unvisited room or the first uncompleted interactive object within the first lesson on the list. If the first lesson on the list still requires completion of a lesson, then the requirements of the lesson are searched for room or object. Thus, the student is advised to make progress in the first lesson on the list, or one of its subordinate lessons.

If, instead, the student is inside a lesson, then the current lesson's rooms and subordinate lessons are searched first and the advice of the aimless agent will be to complete the next requirement from the list of the current lesson.

9. AGENT EXPERIENCE

The fall semester of 2005 saw the first use of these agents by students in a class. The authors were confident of the potential but experienced numerous implementation problems. These implementation problems, when they were manifested, did not have a large impact on the student perception of ProgrammingLand. The usual symptom was that the agent did not appear. Since agents appear rather unpredictably from the student perspective, their absence was not a problem. However, the researchers lost quantitative data.

The subjects of this study were two introductory programming classes at two different institutions; one used Java and the other C++. Although both classes used ProgrammingLand as primary means to deliver content with no other textbook, they each had their own unique approach. In the class using C++, ProgrammingLand used goalie-delivered assignments. Since the student is unable to receive an assignment until completion of a lesson, there is a pressure to use the system. The Java class did not have that pressure, since it did use that technique to mandate use of ProgrammingLand.

The students' subjective views were mostly positive. The good reports from students started rather quickly. There is no learning curve for a college student using a textbook, but there is one for ProgrammingLand and the lost and aimless agents seem to help this problem. However, the responses were not all positive. Students also found the agents to be annoying at times, so more work needs to be done in this area.

The quantitative data is partial, due to the technical difficulties. The logging for the lost agent was completely absent. The aimless agent provided what appears to be reasonable data in September and November and the following analysis is based on this partial data.

The aimless agent made 53 visits to students that appeared to not be making progress towards the goal. The bulk of these (41) were in the C++ class, even though the Java class had more students. This seems to be due to the pressure to complete lessons. When the aimless agent appears, it offers to transport the student to a location that may be more helpful. The students accepted the offer more than 70 percent of the time for both groups. This does not appear to decline significantly over time. This appears to be significant, for if the help the agent provided was of little value, the acceptance rate should seriously decline. The absence of such a decline indicates that the students did find this a helpful service.

The aimless agent's visit has a correlation of -0.489 with students' test performance in the tests in the C++ class. The expectation is that the better students do not wander aimlessly, so they receive fewer visits. The correlation between accepting an offer to be transported to test performance is -0.636. This indicates that the better students not only receive fewer visits, but accept the offer less frequently. These are the type of results to be desired. The better students will do well no matter what the environment. The role of the agents is to help those who are struggling and the data indicates that these are the students who are being visited. There is no data to support or refute the thesis that the weaker students' scores are increased. However, without any indication that an agent damages a student, the implication is clear: the aimless agent is selectively visiting and transporting the weaker students.

10. SUMMARY AND CONCLUSIONS

The roving goalie agents have been in place for several years and have proven effective in their tasks. The list of equivalent assignments approach has greatly reduced the grossest forms of cheating, since the students' programs do not have exactly the same purpose. It then becomes routine for the program grader to check that the student turned in the assignment that had been given. They have also worked well from a system standpoint, for some time.

The quiz system including the quiz agent and quiz room allows a student to use prior knowledge and experience to finish a lesson more quickly. There are several restrictions that limit the generality of this approach. The student must have fulfilled all of the requirements except visiting one or more lecture rooms. The author of the lecture rooms must also have taken the time to generate suitable questions. If the quiz cannot find at least four questions, it will not offer the quiz and thus the lesson completion cannot be accelerated. Typically there are not that many students who are advanced enough in an area to take the quiz. Most students who try this approach, do so just to avoid more work, often failing the quiz. Even if they are lucky – it is a multiple choice quiz – the educational purpose may not have been served.

The lost agent has addressed a long standing need in ProgrammingLand. Historically, there is little correlation between rooms visited and performance in the course, nor correlation between rooms visited and lessons completed. Sometimes students just wander around. It is the author's expectation that if only the good students wandered out of the appropriate area to satisfy their curiosity then the previous correlations should have been positive. Students sometimes just get lost. The lost detection mechanism seems to work well, but there are some problems.

The aimless agent also fills a need stemming from similar student problems. They wander around their lesson without making progress. The availability of the lesson requirements makes it possible for the aimless agent to give good advice as to the next thing to do.

Student comments have been positive for both ProgrammingLand and the agents therein. The agents tend to reduce the amount of time needed to learn how to use the system. The most frequent negative comment is on the agents arriving too frequently and thus being a nuisance.

Although the quantitative data is at best incomplete due to technical problems in the first semester of use, it does indicate that the agents perform a valuable function in ProgrammingLand. Despite the occasional annoyance, students continue to accept the advice of the agents above the 70 percent level through the duration of the course. The data also indicates that the agents are more likely to assist the weaker students, thus leveling the playing field somewhat.

ProgrammingLand and associated software is covered by Gnu GPL. Interested parties should contact the first author.

11. ACKNOWLEDGMENTS

Our thanks to many students who have used, tested and contributed to this project. ProgrammingLand has been supported by ND-EPSCoR through the FLARE program under EPS-9874802 and is currently supported by National Science Foundation Grant EIA-0313154. The authors gratefully acknowledge both of these organizations.

12. REFERENCES

[1] Curtis, Pavel, Mudding: Social Phenomena in Text-Based Virtual Realities. *Proceedings of the Conference on Directions and Implications of Advanced Computing* (sponsored by Computer Professionals for Social Responsibility)

[2] Hill, Curt, Brian M. Slator, Lisa M. Daniels. The Grader in ProgrammingLand. In *Proceedings of SIGSE Technical Symposium on Computer Science Education (SIGCSE 2005), St. Louis MO.,* (March. 2005).

[3] Hill, Curt, Brian M. Slator and Lisa M. Daniels. An Online Resource for the Introductory Programming Class. In *Proceedings of the 2nd International Conference Information Technology Research and Education (ITRE '04)* (London Metropolitan University, London, UK, June 28 -July 1, 2004). IEEE, Piscataway, NJ 08855-1331.

[4] Hill, Curt, Vijayakumar Shanmugasundaram and Martina Miteva. Database Tools to Administer Programming Land. In *Proceedings of 18th International Conference on Computer Applications in Industry and Engineering (CAINE 2005)*. Honolulu, HI., (November 2005).

[5] Holmevik, Jan Rune, and Cynthia Haynes. encore, Open Source MOO project. http://lingua.utdallas.edu/encore/index.html Date accessed 9 January 2006.

[6] Saini-Eidukat, B., Schwert, D.P., Slator, B., Daniels, L., and Terpstra, J., 2005, Research on authentic assessment using a virtual world for learning geology. Geological Society of America North-Central Section, 39th Annual Meeting, 19-20 May 2005, Minneapolis, MN.

[7] WWWIC (2005). World Wide Web Instructional Committee. http://wwwic.ndsu.edu/. Date accessed 9 January 2006.