Pre-Games: Games Designed to Introduce CS1 and CS2 Programming Assignments

Ray Giguette
Computer Science Department
Nicholls State University
Thibodaux, Louisiana 70310
giguette@msn.com

Abstract

Many CS1 and CS2 instructors have been using game-like assignments and manipulatives to increase student understanding of abstract concepts. A related approach described in this paper requires students to play a game-like version of each assignment (referred to as the "pre-game") before designing and coding their own programs. Students complete a prototype of the pre-game that uses simpler I/O but otherwise follows the same specification. By playing the pre-game, students become familiar with lesson principles using a concrete example that combines the advantages of both game-play and manipulatives. Ideally, the pre-game creates an environment that both encourages experimentation, and allows students to use their intuition when designing algorithms and data structures.

Categories and Subject Descriptors

K.3 [Computers and Education]: Computer and Information Science Education – Computer Science Education.

General Terms

Algorithms, Design, Experimentation.

Keywords

CS1, CS2, Data Structures, Visualization, Pedagogy.

1 Introduction

Various methods have been used to create more interesting CS1 and CS2 assignments. Examples include assigning game-like programs [3,4,6], programs simulating real-world data structures (e.g., simulating operating system queues) [2], or introducing concrete examples of data structures, such as binary trees made from PVC pipe [1].

However, certain factors may limit the usefulness of these methods. Though using hands-on manipulatives can increase student understanding of abstract concepts [5], they tend to be practical only for the simplest concepts and structures. It may be difficult, for instance, to invent manipulatives representing structures such as heaps or hash tables, or dynamic concepts such as tree traversal or recursion.

And though students have no trouble playing games, they often have trouble programming them. Implementing realistic details, flexible user interfaces, or interesting graphics is beyond the capabilities of many CS1/CS2 students. To keep assignments doable, complexity unrelated to the lesson must be minimized. For instance, text output must be substituted for graphics and animation. Consequently, game assignments often resemble early versions of Pong or Adventure, rather than the CD-ROM games attractive to students.

The approach introduced in this paper is to have students play a game-like version of each assignment (referred to as the "pre-game") before designing and coding their own programs. Students are assigned to implement a prototype of the pre-game that uses simpler I/O and graphics but otherwise follows the same specification. The pre-game may therefore freely incorporate interaction, animation, user interfaces, etc. that enhance its playability. By playing the pre-game, students become familiar with lesson principles using an interesting, concrete example that combines the advantages of both game-play and manipulatives.

2 Common Student Problems

Instructors are experimenting with game-like assignments and manipulatives in an attempt to address common problems encountered by many CS1 and CS2 students. A few of these problems are listed below:

1. **Following instructions**. Students have trouble following written instructions, often ignoring or misreading important details of program requirements. Some depend on their perhaps incomplete memories of class lectures, rather than re-reading the assignment. Even class discussions are often inadequate, if they consist of notes and diagrams drawn on a black board. To young learners, especially those visually or manually oriented, a static, written description does not fully capture the dynamic qualities of an algorithm or data structure.

Permission to make digital or hand copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, require prior specific permission and/or a fee.

SIGCSE 2003, February 19-23, 2003, Reno, Nevada, USA.
Copyright 2003 ACM 1-58113-648-X/03/0002…..$5.00

2. **Working through problems**. Too often, students faced with a programming quandary are unwilling to work through the problem. They immediately stop and ask for assistance, or just stop altogether. Yet nothing is more valuable than the experience of working through such problems, if possible, unassisted. Students' unwillingness to push ahead is due partially to their resistance to experimentation. Viewing the results of minute alterations to a binary tree is not the average student's idea of a good time. They become frustrated, for instance, comparing a tree stored as a list or array to a tree-like representation drawn on paper. Though students must learn to accomplish such tasks, an environment that facilitates experimentation would be helpful.

3. **Designing algorithms.** CS1 and CS2 students often do not understand what an algorithm is. Many are convinced that an "algorithm" must be something cryptic and unfamiliar. Others mistake the problem statement or the program code for the algorithm. Students therefore have difficulty designing algorithms, and cannot understand why a design is even necessary. These problems may be alleviated if the algorithms could be presented in a different manner, one less abstract and more intuitive.

To summarize, it would be helpful to reinforce the static, written problem statement with a dynamic, visual, one, and to create a programming environment that both encourages experimentation and allows students to use their intuition when designing algorithms and data structures.

3 Advantages Of Pre-Games

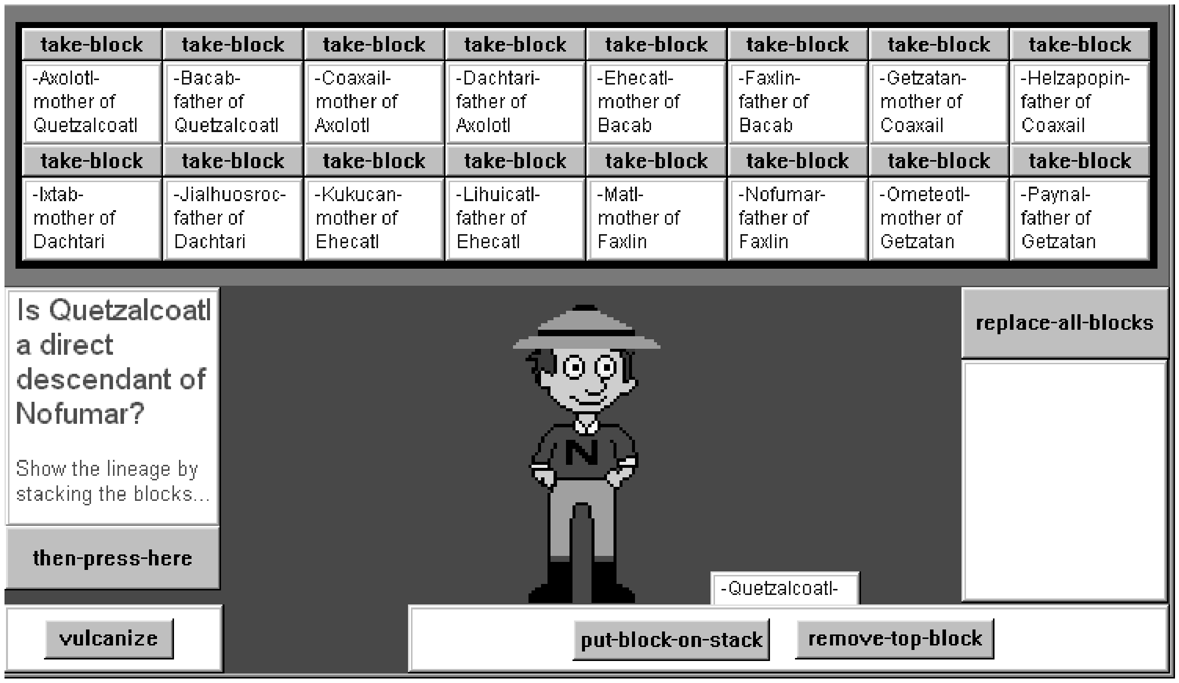
The problems listed above are not new, but hopefully the use of Pre-games represents a new approach to addressing them, by doing the following:

1. **Presenting concrete examples.** Pre-games have many advantages of animated algorithms, while being less abstract. Using a data structure in a game makes it more familiar and intuitive. Using a mouse to manipulate a data structure on a computer screen encourages students to learn both manually and visually. The pre-game, which, except for its I/O follows the same specification as the programming assignment, acts as a dynamic, visual version of the problem definition.

2. **Providing an environment for experimentation.** Students expect games to be challenging; they do not quit when they encounter a roadblock, because these "roadblocks" are what make the game enjoyable. It is hoped that students will adopt this mindset when playing a pre-game. If a pre-game presents an environment in which new ideas and strategies can be easily tested, students may spend more time working through problems before seeking help.

3. **Regarding algorithms as "strategies"**. Equating an assignment with a game encourages students to design algorithms in two stages:
   a) First, define the rules of the game.
   b) Second, define a winning strategy.

   For instance, the algorithm for a solitaire Tic Tac Toe program must both enforce the rules of the game and implement a strategy for making computer moves. While the first part is clearly defined and relatively simple, the second is less so, and allows students to develop their own ideas. Students may have to play a game a few times to fully understand it; what works in one case may not work in all cases. Students can thus compare strategies that are always successful, to ones that succeed most of the time, to ones that usually fail. They may come to understand that designing a general algorithm is similar to developing a winning game strategy.



-Axolotl- mother of Quetzalcoatl	-Bacab- father of Quetzalcoatl	-Coaxail- mother of Axolotl	-Dachtari- father of Axolotl	-Ehecatl- mother of Bacab	-Faxlin- father of Bacab	-Getzatan- mother of Coaxail	-Helzapopin- father of Coaxail
-Ixtab- mother of Dachtari	-Jialhuosroc- father of Dachtari	-Kukucan- mother of Ehecatl	-Lihuicatl- father of Ehecatl	-Matl- mother of Faxlin	-Nofumar- father of Faxlin	-Ometeotl- mother of Getzatan	-Paynal- father of Getzatan

Is Quetzalcoatl a direct descendant of Nofumar?

Show the lineage by stacking the blocks...

-Quetzalcoatl-

Figure 1. Stack Pre-Game Screenshot

4 Pre-Game Examples

The following pre-games were created using the MicroWorlds programming environment[1]. MicroWorlds has built-in graphics, interface, and animation capabilities that increase game playability. These games introduce students to data structures such as stacks, graphs, and trees, and their related algorithms. They are presented as individual assignments connected by a story involving the lost treasure of the Aztecs. In each pre-game, the student plays the part of a treasure-hunting explorer.

4.1 The Stack Pre-Game

The explorer finds himself in a room with two shelves containing 16 stone blocks, each bearing the name of a different Aztec god (Figure 1). For instance, one block is labeled "Bacab: father of Quetzalcoatl". Another block, labeled "Quetzalcoatl", lies partially concealed in a spring-loaded recess in the floor. The explorer discovers that he can pick up shelved blocks and place them one at a time on top of the Quetzalcoatl block. However, the weight of each extra block causes the stack to sink further into the floor, so that only the top block is visible at any time.

The explorer's job is to stack some of the shelved blocks on top of the Quetzalcoatl block so that they trace the lineage from Quetzalcoatl to his ancestor Nofumar. This is accomplished by first placing one of Quetzalcoatl's parents on the stack, then a parent of that parent, and so on, working backwards until the Nofumar block is on the top of the stack.

If at any step the explorer selects the wrong ancestor, he will eventually reach a dead end. He must therefore develop a strategy for adding and removing blocks, and keeping track of which blocks have already been tried. (If the explorer is clever, he can use a nearby pool of lava to dispose of blocks removed from the stack.)

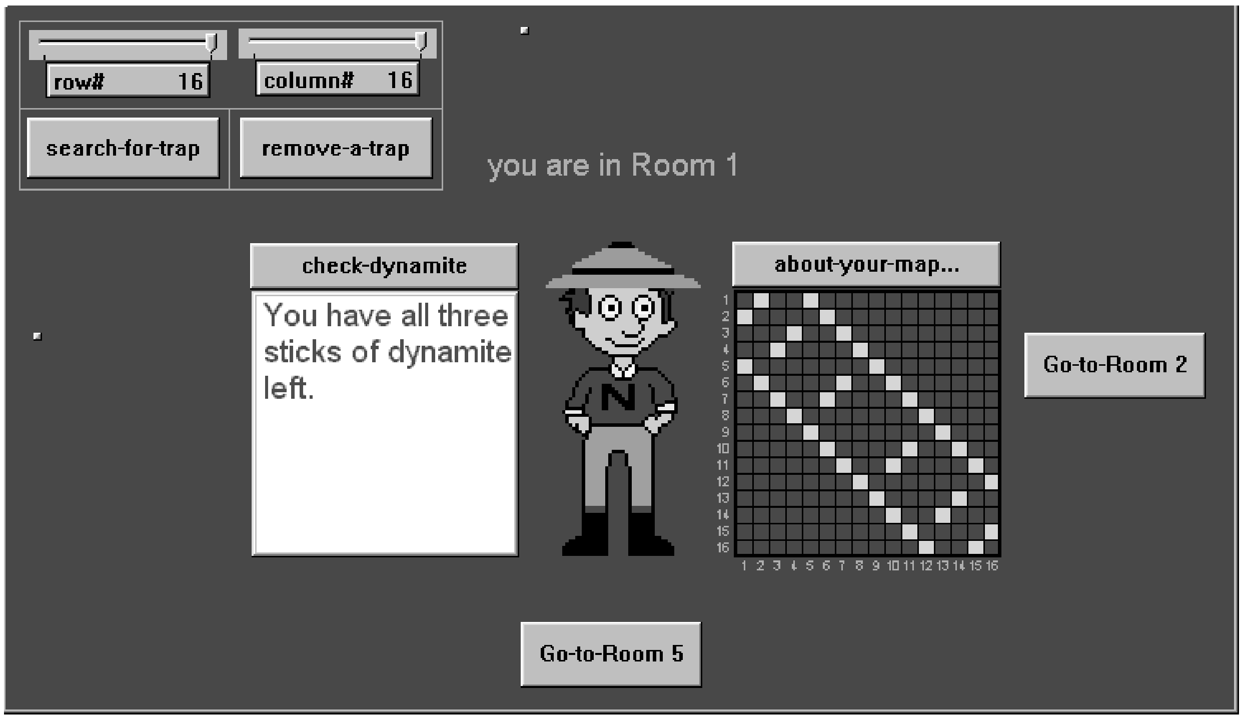
The game allows the student to manipulate a stack and experiment with backtracking. Some students may "cheat" by looking ahead to find the correct sequence of gods. It can then be pointed out that this strategy would be difficult to program efficiently, especially for 10,000 blocks instead of 16. Instead the student must invent a general strategy that works for any set of gods, and takes into account the possibility that there may be no direct lineage between two arbitrary gods.

The need for a general strategy would be more apparent if a random set of god's was created each time the game was played. However, because this would increase the difficulty of the assignment, it is better left as an extra-credit option.

4.2 The Graph Pre-Game

The explorer finds himself in the entrance to a 16-room maze (Figure 2). The rooms are numbered; he is in room 1 and he must find room 16. Each room connects to up to four other rooms (to the north, south, east, and west). Some of the doors have been booby-trapped, so he must examine each door before going through it. If he finds a trap, he can remove it using a stick of dynamite.

The explorer has a map and three sticks of dynamite. His map appears to be a checkerboard, but is actually a color-coded 16 x 16 adjacency matrix. I.e., the square at row j, column k represents the door between rooms j and k. Initially, all map squares are



You have all three sticks of dynamite left.

Figure 2. Graph Pre-Game Screenshot

yellow, indicating that the doorways have not yet been searched for traps. When a doorway is searched, the map-square turns either green (if no trap is found) or red (if a trap is found).

[1] MicroWorlds is a registered trademark of Logo Computer Systems, Incorporated.

To play the game, students must be able to read and modify the adjacency matrix. To examine a doorway, the explorer sets two values, using two sliders, representing the row and column. To remove a trap, the explored must likewise indicate the row and column of the doorway. Removing a trap uses up one stick of dynamite.

The strategy for this game is not as obvious as in the stack pre-game, and students are encouraged to come up with their own strategy. Initially, they may wander through the maze, arbitrarily removing traps. Because the number and position of traps is randomly determined, the game might be played a few times before students encounter the "worst case" scenario, which forces them to find the shortest path through the maze in order to minimize dynamite use.

As an extra credit assignment, the difficulty of the game can be adjusted by altering either the number of traps or the amount of dynamite. However, if too many traps are created, even the best strategy will not work.

4.3 The Binary Tree Pre-Game

The explorer finds himself in front of a door with the sign "ATM[2] inside" (Figure 3). He is instructed to open the door to the treasure room by inputting his PIN, a 4-letter code.

On the floor are two tiles: one black the other white. These sink into the floor when stepped on, then pop up when released. (The student uses the mouse to select a particular tile.) When a tile is depressed, the corresponding letter (either B or W) appears in a display. In this way, a sequence of Bs and Ws may be entered. There is also a "back up" key, which erases the last letter entered.

To enter his PIN, the explorer must step on the tiles in a particular sequence. For instance, one possible sequence is B, W, B, B. Unfortunately, the explorer does not know his PIN, so he must enter every possible permutation of Bs and Ws of length four until he finds the one that opens the door. He can do this the hard way, by "manually" entering all 16 4-letter sequences, keeping track of which sequences have been tried, and which have not. Or he can use the binary tree attached to the wall to create and test all possible 4-letter sequences.

Each edge of the tree is a wooden ledge, and each node is a button labeled either "B" or "W". A node's label can be toggled by clicking on it. Initially, all nodes are labeled "W".

The tree has four levels. At the top of the tree is a boulder which, if pushed, will travel up and down the tree ledges in a pre-order traversal. Each time the boulder rolls *down* a ledge to a node, that node is displayed. Each time the

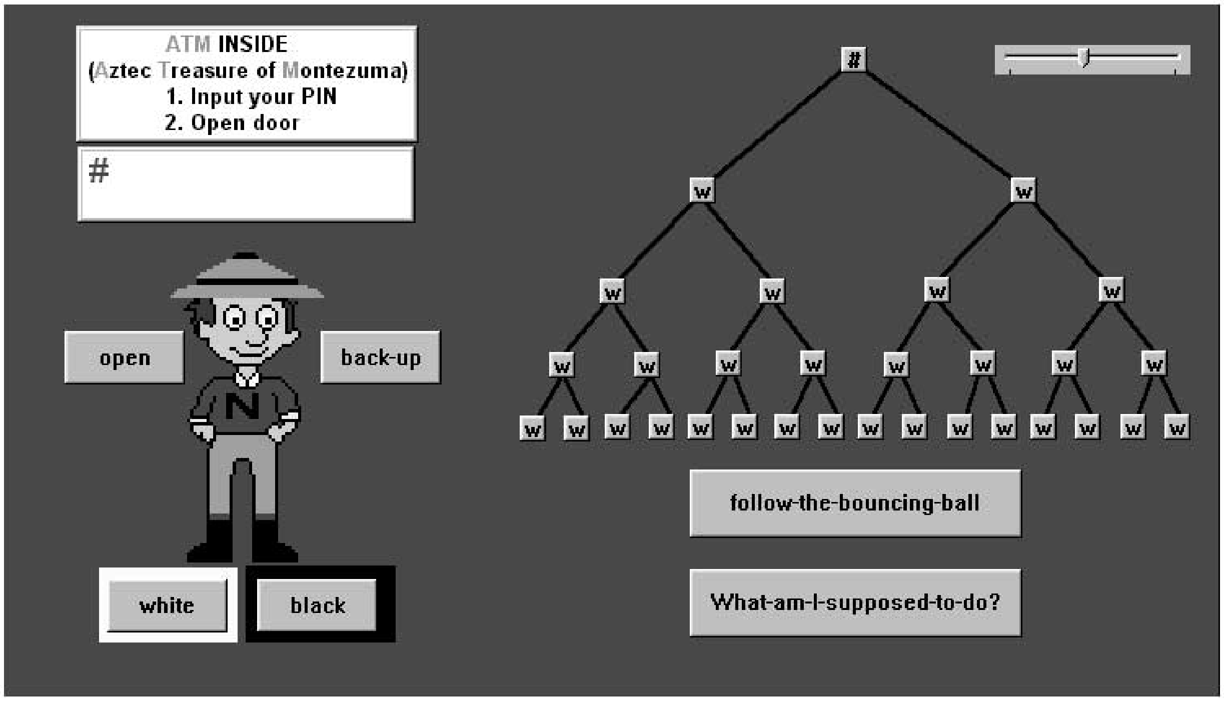

ATM INSIDE
(Aztec Treasure of Montezuma)
1. Input your PIN
2. Open door
#

Figure 3. Binary Tree Pre-Game Screenshot

boulder bounces *up* a ledge, the last letter is erased from the display. Whenever the boulder reaches a leaf node, the

[2] Aztec Treasure of Montezuma

4-letter sequence appearing on the display is automatically tested to see if it opens the door.

Rolling the boulder thus causes sequences of Bs and Ws to be entered into the display. The sequences formed depend on the values of the tree nodes. Initially, because all the nodes are set to W, the sequence "WWWW" is entered repeatedly. The explorer must determine how to change the node values so that a pre-order traversal of the tree will form all possible 4-letter sequences. To encourage a general solution, a random PIN is selected each time the game is played.

The solution is relatively simple (e.g., assign each left child a "W" and each right child a "B"). However, students often have difficulty manipulating a tree to achieve this solution in a normal programming environment. The pre-game allows students to more easily see the result of modifying the data stored in a binary tree.

A strategy sometimes used by students is to randomly change node values until, by luck, a traversal creates the correct PIN value. However, students are warned that this strategy will prevent them from succeeding in the next part of the pre-game, in which they must implement the recursive algorithm needed to traverse the tree.

5 Conclusion

It is hoped that presenting CS1 and CS2 concepts in the form of a game will make students more receptive and work harder. An underlying assumption is that the more interesting the pre-game, the more effective it will be. The ultimate pre-game would be as enjoyable as a "real" game, and could only be won by understanding its underlying algorithms and data structures.

Pre-game playability can be enhanced as long as any resulting increase in complexity is omitted from the student assignment. This difference in complexity between the pre-game and the corresponding assignment allows students to select their own level of achievement. Average students can complete the basic assignment, while superior students can program a more complete prototype, by incorporating better strategy or I/O, or by modifying the game rules.

The pre-games presented in this paper are relatively simple and relatively new, and their ultimate usefulness is a subject of further research.

References

[1] Bucci, P., Long, T., Weide, B., and Hollingsworth, J. (2000). Toys Are Us: Presenting Mathematical Concepts on CS1/CS2, *Proceeding of the Frontiers in Education Conference*, Kansas City, Missouri, F4B1– F4B6.

[2] Jimenez-Peris, R., Khuri, S., and Patino-Martinez, M., (1999). Adding Breadth to CS1 and CS2 Courses Through Visual and Interactive Programming Projects, *Proceedings of the 30th SIGSCE Technical Symposium on Computer Science Education,* New Orleans, Louisiana, 252-256.

[3] Liss, I., and McMillan, T. (1988). An Amazing Exercise in Recursion for CS1 and CS2, , *Proceedings of the 19th SIGSCE Technical Symposium on Computer Science Education*, Atlanta Georgia, 270-274.

[4] Reese, D. (2000) Using Multiplayer Games to Teach Interprocess Communication Mechanisms, *SICSCE Bulletin*, 32, 4, 45-47.

[5] Resnick, F., Martin, F., Berg, R., Borovoy, R., Colella, V., Kramer, K., and Silverman, B. (1998). Digital Manipulatives: New Toys to Think With, Conference *Proceeding on Human Factors and Computing Systems*, Los Angeles, California, 281-287.

[6] Stone, D., and Schmalzel, J. (1999). A CS1 Maze Lab, Using Joysticks and MIPPETs, *Proceedings of the 30th SIGSCE Technical Symposium on Computer Science Education,* New Orleans, Louisiana, 170-173.