Puzzles and Games: Addressing Different Learning Styles in Teaching Operating Systems Concepts

John M. D. Hill, Clark K. Ray, Jean R. S. Blair, and Curtis A. Carver, Jr.

Department of Electrical Engineering and Computer Science

United States Military Academy

West Point, NY 10996

{John.Hill, Clark.Ray, Jean.Blair, Curtis.Carver}@usma.edu

Abstract

Because students have different learning styles, it's important to incorporate multiple teaching techniques into the classroom experience. One such technique is the use of puzzles and games in the classroom to reinforce the learning objectives. Many topics in Computer Science are well suited for coverage in such a game. Several in-class puzzles and games have been used in the Computer Science program at this institution in recent years. In basic and advanced courses, simple crossword puzzles reinforce terminology and *Jeopardy!*®-style games help students master material with short answers. In the most recent iteration of the *Operating Systems* course, a *BattleThreads* game and a *Process State Transition* game helped students appreciate different approaches to process and thread management. The latter two games have been assessed for their effectiveness, providing several insights into what makes a good in-class game for teaching operating systems concepts, and how the existing games can be improved.

Categories and Subject Descriptors

K.3.2 **[Computers and Education]** Computers and Information Science Education - *Computer Science Education*.

General Terms

Algorithms, Experimentation, Human Factors

Keywords

Learning Styles, Classroom Games, Operating Systems

1 Introduction

It's not a big secret that different students have different learning styles. For example, Felder's model includes several basic dimensions: sensory/intuitive, visual/verbal, active/reflective, and sequential/global. [3] This means that teachers can reach more students by using a variety of instructional techniques. In addition to the textbook, the lecture, the assignments, and even online course resources, it's useful to incorporate hands-on exploration into the classroom. This can be an important component of using learning style theory to improve engineering education. [13]

SIGCSE'03, February 19-23, 2003, Reno, Nevada, USA.

Because any collection of students embodies many different preferred learning styles it is useful to incorporate multiple teaching strategies, including in-class games. The use of games, particularly competitive games, as an additional teaching tool in the classroom is not a particularly new idea. They are commonly used in economics and social science classrooms [6], in teaching mathematics [1], and to support science courses. [5]

With the availability of modern programming and prototyping tools, many useful simulations and web-based applications have been added to the normal teaching techniques. However, Computer Science students in particular already have plenty of "face time" with keyboards and screens. In the Computer Science program at this institution, students at every level from freshmen to seniors have, in several post-course assessments, indicated that they prefer a few more in-class exercises or similar techniques to reinforce the learning objectives. As a result, faculty members are exploring different hands-on experiences, including games that can be used to teach and reinforce Computer Science concepts.

2 Related Work

There are several people using simulations, web-based applications, and games in support of teaching Computer Science concepts. Tim Bell describes how to present fundamental Computer Science ideas to general audiences. [2] Ohlsson and Johansson have used role-playing games and practical exercises for software engineering education. [8] In the realm of operating systems courses, Robbins and Robbins report on using a process scheduling simulator to support hypothesis-testing [10] and another simulator for the examination of synchronization. [11] The 2001 SIGCSE conference inspired the creation of a maze demonstration program suitable for student exploration of a maze traversal algorithm. [9] Levitin and Papalaskari examined the use of puzzles to illustrate the operation of some routinely-taught algorithms (brute-force search, divide-and-conquer, and other strategies). [7]

3 Puzzles and Games for any CS Course

Crossword puzzles and *Jeopardy!*®-style games were recently

The views expressed are those of the authors and do not reflect the official policy or position of the United States Military Academy, the Department of the Army, the Department of Defense or the United States Government.

used in several Computer Science courses at this institution ("*Jeopardy!*" is a registered trademark of Jeopardy Productions, Inc). The instructional crossword puzzles and *Jeopardy!*®-style games described below are examples of ways to create in-class experiences that support learning objectives involving terminology and basic concepts.

3.1 Crossword Puzzles

Students in a senior-level *Fundamentals of Computer Theory* course were offered *Crossword Puzzles* as a means of reinforcing definitions of terms. Not all of the students were required to complete the *Crossword Puzzle*s. Rather, some students completed them as an in-class exercise with immediate feedback from the instructor and other students completed them outside of class. The *Crossword Puzzles* are designed more for knowledge reinforcement than for discovery of new concepts. They are examples of individual games that do not allow for competition between the players (other than time to complete or amount correct).

3.2 The *Jeopardy!*® Game

In many courses there is a large amount of information that falls into the "know" or "be familiar with" or "know how to" categories. Definitions of terms are one example, as are simple problems, calculations, or algorithms. These are ideal candidates for inclusion in a *Jeopardy!*®-style game.

Students in a freshman-level *Introduction to Computer Science* course and a senior-level *Fundamentals of Computer Theory* course were presented with a *Jeopardy!*® game tailored to the knowledge appropriate to the course. The game is implemented in HTML and requires at least an instructor workstation and a means of projection. If placed on a publicly available server, students can access it for exploration at their own pace. However, this game works best in the classroom where students compete with each other and scores can be kept.

The columns of the board are populated with topic areas drawn directly from the learning objectives. Figure 1 shows a layout for context-free grammars and pushdown automata from the *Fundamentals of Computer Theory* course.

REGULAR	NOT	READ CFG	DESIGN CFG	CONVERT	PDA
200	200	200	200	200	200
400	400	400	400	400	400
600	600	600	600	600	600
800	800	800	800	800	800
1000	1000	1000	1000	1000	1000

Figure 1: Double *Jeopardy!*® Layout for Context Free Grammars and Pushdown Automata

The game board can be set up with the answers like actual *Jeopardy!*®, but is typically set up with questions. Some examples are shown in Figure 2. Feedback on the correctness of student answers can be automated or can come from the instructor. Another advantage to performing this game in class is that the instructor can provide explanations or additional comments.

Convert the following grammar to Chomsky Normal Form.	Construct a grammar whose language is $\{a^m b^n c^p d^q\}$
$S \rightarrow SS$ $S \rightarrow (S)$ $S \rightarrow e$	where $(n = q)$ or $(m <= p)$ or $(m+n = p+q)$

Figure 2: Sample Questions for CFG and PDA

3.3 Assessment of Crosswords and *Jeopardy!*®

The students provided positive remarks when asked their opinion of the crossword puzzles, stating that they were an effective technique for reinforcing knowledge of terminology. The *Jeopardy!*® game was also generally well received. However, several students commented that at any particular moment only a few students were actively engaged. One student offered the suggestion that the game should be played in smaller groups, perhaps even in different classrooms if available.

4 Games for the *Operating Systems* Course

The two games described below were developed in response to assessment results from second-semester juniors at the end of the *Design and Analysis of Algorithms* course. They indicated that they would "prefer more in-class practical exercises" because they were "not willing to invest practice time unless forced to do so." Their comments suggested that in-class activities aimed at students who were more visual / sensory / global learners might be appropriate. The games described below were developed to support the same body of students as they became first-semester seniors in the *Operating Systems* course. The descriptions of the games and the assessment of their effectiveness by the students and faculty serve to illustrate how course concepts can be converted into games, how the games can be run, and how they might be improved in the future.

4.1 The *BattleThreads* Game

Students in the senior-level *Operating Systems* course must come to an in-depth understanding of the algorithms used for process management, prioritization in scheduling, memory management, input/output control, and many other topics. Some in-class games were developed particularly in response to course assessment responses from this population of students, who stated that they would prefer more hands-on in-class exercises to reinforce complicated material.

One of those games was a modification of the well-known BattleShip™ game (BattleShip is a registered trademark of the Hasbro Corporation). One iteration of this game was used to demonstrate the differences between processes and threads, and the advantages of communication between threads in the same processes through the shared address space. This game was designed to support the following learning objectives:

- "Students will know what threads are and the distinctions between threads and processes."
- "Students will understand the advantages of a multithreaded organization in structuring applications and in performance."

For this game, the class is broken down into one controller and some number of players for each of two teams. The dimensions of the game board and the size of the ships are tailored to the number of student players (to raise the probability of a hit). An optional rule requires that ships cannot be adjacent, even on the diagonal. The players are responsible for the placement of one ship each, and for firing a shot from that ship each turn until their

own ship is destroyed (or their side wins by sinking all of their opponents' ships). The controllers get the enemy team's ship layout and tell all of their own players the effect (hit or miss) of each shot. At the end of the turn, the controllers compare battle damage and report the results back to their players.

As an example, consider twelve students broken down into two teams with five players and one controller each. For this scenario a seven-by-seven grid is sufficient if the ships are limited in size to occupy exactly three grid squares. This also means the ships can be destroyed pretty quickly when found, making the game flow faster. The two teams were given different rule sets. One team represented a single process broken down into five threads, all with access to the same address space. This address space includes the array that represents the enemy grid. Any change to the array (the recording of a hit or miss) was immediately visible to all players. No other means of communication was allowed. The other team represented individual processes that could only communicate using inter-process communications mechanisms. This was modeled in a rule that allowed them to either take a shot or to communicate the effect of the last shot to the rest of the processes. The two teams were not informed that they were working with different rule sets.

Once the game began the differences in the two approaches rapidly becomes apparent. In the threads team, as soon as a hit was recorded, the following players could target the wounded ship until it was sunk. On the process team, the players had no information about what the other players had done unless one chose to announce a result instead of taking a shot. When a hit was announced, several players would often try to destroy the same ship, resulting in multiple strikes on the same location.

Figure 3, below, shows the ships as shaded cells and demonstrates the status a game at the conclusion of the fourth turn. The ability of the threads team to target a wounded ship for destruction is evident, as are the multiple strikes (M2) of the process team. Although the procedure appeared to be the same for both teams, the process team rapidly caught on that the threads team was able to communicate more effectively and fire more accurately.

Threads – Enemy Board

	1	2	3	4	5	6	7
A							H
B	H	H	H	M		M	
C	M				M		
D	M	H		H	H	H	
E		H	M	M			M
F		H					
G			M		M		

Process – Enemy Board

	1	2	3	4	5	6	7
A	M						
B		M			M	M2	H
C			M2	H			
D		H		M		M	
E					M	M	
F	M	M	M				
G			H	M		M	

Figure 3: Enemy Boards for the Threads Team and the Process Team

Several different approaches can be used starting with the same basic game. It is important to note that many of the details of inter-process communication and shared memory were initially suppressed for ease of play. Once the students become familiar with the basic play, the rule set can be changed to emphasize different concepts. As an example, an alternate approach was used for one class in which the instructor guided the students as they developed their own rules. In this approach, an extensive interactive discussion of process and thread issues took place, followed by modification of the rules to reflect various process and thread management strategies.

4.2 The *Process State Transition* Game

In an *Operating Systems* course that uses the textbook by William Stallings [12] a major topic involving process management was supported by the creation of a *Process State Transition* game. According to Stallings, one of the most important achievements in the history of operating systems design is the concept of a *process*. His discussion of processes begins with an examination of models of process states. One of the most important functions of the operating system is the management of processes, which can be represented as just such a state model. This sets the foundation for future discussion of how processes are described (operating system control structures, process control structures, etc.) and how they are controlled (modes, creation, switching, etc.).

For this particular game, the supported learning objectives were:

- "Students will understand how to model process management as the transition of processes between execution states."
- "Students will understand the data structures maintained by the operating system to manage processes."
- "Students will understand how the operating system performs the scheduling function."

The members of the class are broken down into sets of 4-6 students. Each set of students is given a game board representing the seven-state process transition model used in the Stallings textbook (see Figure 4). One of the students is selected to be the operating system (OS), one is selected to be the timekeeper (TK), and the others become programs, each keeping track of some number of processes as they are managed by the operating system.

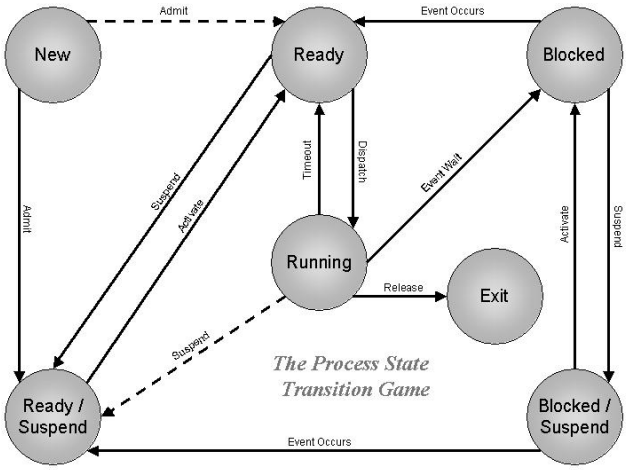

Figure 4: Modified Figure 3.8(b) from the Stallings *Operating Systems* Textbook

Processes are represented as a sequence of *processing time* (a capital P followed by a number indicating the number of time units used for processing) and *blocked time* (a capital B followed by the number of time units. See the example in Figure 5 below.

PROCESS A1: P3, B4, P2, B5, P3
PROCESS A2: P2, B3, P3, B2, P1
PROCESS B1: P3, B5, P1, B3, P2

Figure 5: Process Representation

It's important not to assign so many processes to the players that the game gets bogged down, or so few processes that there is no need for processes to be suspended.

Each process has an associated memory requirement. When a process moves into a state in which it must be present in memory (Ready, Running, etc.), the player who owns that process places its memory markers on a grid representing available memory (see Figure 6). When the process is suspended, the memory markers are lifted, indicating that the process has been moved out of main memory (usually to make room for another process).

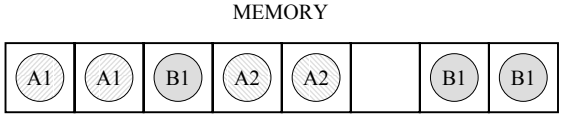

Figure 6: Allocation of Main Memory to Processes

As the game progresses, students discover the complex management issues associated with the management of processes, including deciding which processes to suspend, activate, or dispatch, and how to queue and prioritize processes.

An alternative approach to playing the game is to send student groups to the chalkboards instead of working at desks. The students draw and manage memory slots, CPU utilization, the process state diagram, the timeline, and process control blocks. Rules for processes are issued to each student assigned a process role. In this format, all groups are permitted and encouraged to view the other groups' boards. The game turns move synchronously at the direction of the instructor, and at the end of each turn all groups explain their result and discuss (and sometimes argue) the differences in their results. In this way, it is easier for students across groups to confer on proper outcomes and on the most realistic interpretation of rules, and to compare the results at the end of each turn. Controlled arguments across groups as to 'who is right' or 'what is more realistic' seemed to prove both informative and engaging.

4.3 Student Assessment of *Operating Systems* Games

Following the administration of the *BattleThreads* and *Process State Transition* games, the students were asked to respond anonymously to formal self-assessments about their experience with the games. The assessment questions were tied directly to the learning objectives or to the overall effectiveness of the game. For example, students were asked to respond on a scale of "Strongly disagree" through "Strongly agree" to statements like "The *Process State Transition* game helped me to understand what causes a process to transition between states."

The self-assessment results for the *BattleThreads* game showed that the students reacted positively; a large majority of the students favored this approach to mastering concepts. They felt they were able to rapidly grasp the distinction between threads and processes and the advantages and disadvantages of different types of communication. The students' perceptions no doubt benefited from the ability to get started quickly with a familiar game format (BattleShip™) and the stark contrast in the efficiency of the two different rule sets.

One important observation is that the cost (in terms of time invested in game issues) versus the benefits (learning) ratio may have been higher using the second approach in which students determine their own rules. Conceptual issues occupied almost all of the time spent on game (rule) discussions, even though the students were tapping into a familiar knowledge setting (the BattleShip™ game) .

The self-assessment results for the *Process State Transition* game were good. Most of the students agreed or strongly agreed that the game was better than covering the same material in a lecture, and that it helped them understand the concepts. However, there were a small number of students who disagreed or strongly disagreed. This suggests that the game format was not appropriate for those particular students' favored learning style.

One advantage to the alternate implementation of the *Process State Transition* game was that the boards were more visible and there was more room for the groups to work. There is also a disadvantage, in that working at the boards is more familiar and less distinctive in setting and sensation than working with game components around desks. As a result, students may lapse into less receptive 'at the board' mode. In this approach, students in the role of processes seemed less engaged than in the straight game-playing mode. However, this approach allowed students to take notes on what they were doing, an activity that should have been encouraged even in the game-playing approach, but wasn't.

Two very clear points came out of the use of both approaches. The first point is that several students legitimately objected to the presumed negligible cost of context switches, state transitions, and virtual memory operations. The explanation that it was a simplification required to control game complexity was understood but not satisfying. This was actually a good sign, as it indicated the students had begun to understand some of the more complex issues in process management. The initial *BattleThreads* game, in particular, was probably over-simplified.

The second point is that it would be beneficial to provide the game rules as a read-ahead. This provides greater initial familiarity at the beginning of class and reduces the amount of 'discovery' learning as students try to determine how to execute their role by asking peers and comparing disjoint information. This was particularly evident for the *Process State Transition* game, where many of the student groups got bogged down in learning the game rather than in understanding the concepts it was supposed to teach.

4.4 Faculty Assessment of *Operating Systems* Games

To determine the individual learning style profiles, each student was administered the Index of Learning Styles questionnaire. It's important to remember that "the results provide an indication of an individual's learning preferences and probably an even better indication of the preference profile of a group of students (e.g. a class), but they should not be over-interpreted." [4] The assessment of the effectiveness of the in-class games focuses more on groups than on individuals. Although the two instructors themselves had noticeably different learning style profiles (one with central rankings, the other significantly more visual and sensory), there was no appreciable difference between the performances of their sections.

To evaluate student mastery of the two topics covered by the use of in-class games (processes and threads, process state transitions) two questions on those topics were included on a major test. Less than half of the first question was tied to the learning objectives on process and thread management issues covered by the *BattleThreads* game. The second question was completely related to the learning objectives covered in the *Process State Transition* game. The students were given the option of doing either

question. In hindsight, assessment would have been better served if both questions were required. There were no directly comparable questions in graded events from previous years.

Those students who chose the process state transition diagram question performed significantly below the average score for all questions. The use of the *Process State Transition* game didn't seem to help student performance, and may have hurt by denying them the same amount of lecture time. Perhaps the students didn't get enough game time to fully understand all of the transitions. Also, it might have made sense to assign jobs based on learning styles - a global, active, sensory student might make a better operating system, and a sequential, reflective student a better process.

Those students who chose the thread management question performed at the average score for all questions. The use of the *BattleThreads* game did not appear to help or harm accomplishment of the learning objectives.

5 Properties of a Good Computer Science Game

Some of the most useful insights that have come out of the use of games to teach Computer Science topics are the identification of what topics are good candidates for a game, and how the games should be implemented.

The most important point is that the game must be clearly linked to specific learning objectives. Any topic that includes knowledge or definitional items will likely convert easily to word-based games like crossword puzzles and *Jeopardy!*®-style games. Topics that involve clearly defined procedures can usually be turned into a turn-playing game like *BattleThreads* or the *Process State Transition* game. It's necessary to decide early what procedures and issues are being simplified or abstracted for the sake of emphasizing others.

For in-class games, simple implementations are the best. Paper boards, dice, and markers are familiar mechanisms to most students. For games that are intended for use in or out of class, an electronic copy of all of the materials and rules should be made available. If automation support is required (like the *Jeopardy!*® game) then it should be as platform-independent as possible. Special care should be given to synthesizing clear and concise rules that can be learned in a relatively short period of time. Also, play testing is appropriate to ensure the mechanics of operating the game don't overwhelm the learning objective.

6 Conclusions and Future Work

The use of games to teach and reinforce Computer Science concepts based on specific learning objectives has been a positive experience for the faculty and the students. In the process of working with the games, several criteria for game selection and improvement have been determined. The use of games is currently being considered for several other Computer Science topics, and will be implemented, conducted, and assessed to determine their effectiveness. The insights gained on this effort will be used to develop a comprehensive plan for those future efforts. One unanticipated area of future work is that the *Process State Transition* game is now so well defined that it has become a candidate for student implementation in a senior-level simulation course.

In terms of learning styles, one of the biggest challenges ahead is to develop a mechanism for determining which students will respond better to a games approach (recall that some students did not like it at all, and note that although most students liked the *Process State Transition* game they performed poorly when evaluated). Another big challenge is to determine how to provide multiple teaching techniques, including games, without over-burdening the instructor.

References

[1] Begg, A. J. C., "Games in the Classroom," Centre for Innovation in Mathematics Teaching, Available at http://www.ex.ac.uk/cimt/games/gameclas.htm, [September 6, 2002].

[2] Bell, T., "A Low-Cost High-Impact Computer Science Show for Family Audiences," Australasian Computer Science Conference, Canberra, Australia, (January 31-February 3, 2000), 10-14.

[3] Felder, R. M. and Silverman, L. K., "Learning and Teaching Styles in Engineering Education," Engineering Education, Vol. 78, No. 7, (1988), 674-681.

[4] Felder, R. M. and Solomon, B. A., "Index of Learning Styles (ILS)," North Carolina State University, Available at http://www2.ncsu.edu/unity/lockers/users/f/felder/public/ILSpage.html, [September 6th, 2002].

[5] Herr, N., "The Sourcebook for Teaching Science: Strategies, Activities, and Internet Resources," California State University, Northridge, Available at http://www.csun.edu/~vceed002/, [September 6, 2002].

[6] Holt, C., "Y2K Bibliography of Experimental Economics and Social Science Classroom Games - Using Experiments in Teaching," University of Virginia, Available at http://www.people.virginia.edu/~cah2k/classy2k.htm, [September 6, 2002].

[7] Levitin, A. and Papalaskari, M.-A., "Using Puzzles in Teaching Algorithms," Technical Symposium on Computer Science Education, Covington, Kentucky, (February 27 - March 3, 2002), 292-296.

[8] Ohlsson, L. and Johansson, C., "A Practice Driven Approach to Software Engineering Education," IEEE Transactions on Education, Vol. 38, No. 3, (1995), 291-295.

[9] Rasala, R., Raab, J., and Proulx, V. K., "The SIGCSE 2001 Maze Demonstration Program," Technical Symposium on Computer Science Education (SIGCSE 2002), Covington, Kentucky, (February 27 - March 3, 2002), 287-290.

[10] Robbins, S. and Robbins, K., "Empirical Exploration in Undergraduate Operating Systems," Technical Symposium on Computer Science Education (SIGCSE 1999), New Orleans, Louisiana, (March, 1999), 311-315.

[11] Robbins, S., "Experimentation with Bounded Buffer Synchronization," Technical Symposium on Computer Science Education (SIGCSE 2000), Austin, Texas, (March, 2000), 330-334.

[12] Stallings, W., Operating Systems: Internals and Design Principles, Fourth Edition, Upper Saddle River, New Jersey: Prentice-Hall (2001).

[13] Terry, R. E. and Harb, J. N., "Using Learning Style Theory to Improve Learning and Teaching in the Engineering Classroom," Frontiers in Education (FIE 1993), Washington, DC, (November 6-9, 1993), 22-23.