

Games As A Motivation For Freshman Students To Learn Programming

María Feldgen¹ and Osvaldo Clúa²

Abstract - Programming is a difficult skill to acquire. It is best learned by practice and, if students are to learn effectively, at least some of this practice will have to be self-directed. Instructor's key role is to persuade our students to do this and thus to motivate them. In the past, our students identified programming as a vital skill in demand by industry. Consequently they were motivated to acquire a useful skill that would be relevant in some future job or lucrative career. Nowadays, our WEB age students have no idea why they have to study programming. Programming courses are seen simply as mandatory parts of the degree course to be negotiated. Their world of computing is multithreaded computer programs with impressive human interfaces for games and WEB. They cannot relate them with the classical programming exercises that ask for single-threaded programs performing a sequence of calculations. Given this, we decided to introduce our students in problem solving using what they view as real-world problems such as games and WEB-programming.

Index Terms - Freshman, Learning styles, Programming skills, Retention.

INTRODUCTION

The transition from high school to college can be very difficult for many students. Most of the students who drop out of college do so during their freshman year and particularly during the first semester [1]. In the institution where we conducted the study, freshman students have had advising/counseling sessions conducted by a faculty member (tutor) for their first year since 2001. In addition, Math and Physics remedial courses were introduced, as the academic approach to students' retention. The first course's professors and teaching assistants, who meet the students twice a week, do most of the process of mentoring. Tutors relay on these professors' feedback for the analysis of students' behaviors and weaknesses in order to help endangered students to make the transition.

Our 2-semester course of introductory programming is one of these first courses. Students entering our course have no idea why they have to learn programming. They begin to consider it difficult and demanding, but rewarding for their software engineering degree, at the middle of the course. We become popular during the course; students said that we are

introducing them to the engineering practices and share with them the same view of their engineering field. That is not true; we are changing our exercises every year approaching their view of the software engineering profession. We are hiding the same algorithms and mathematical stuff as demanded by the curricula in the type of problems they identify as real-world problems.

Which is the world of computing of our WEB age students? There are games and WEB. This type of problems is motivating for them to search for a solution, is easy to understand (not to solve) and is challenging. They are suitable to work close with the students in the labs, creating a non-threatening environment. Games can be introduced in very different ways, so that we can address the students' different learning styles, different backgrounds and abilities. Many researchers have argued [2] [1] that students who study in ways congruent with their individual learning styles evidence improved academic achievement and increase their ability to exercise control over their own progress. This increases the quality of the learning that should be the guiding principle of retention programs.

MOTIVATION

It is surprising that only a few students are pointing to programming as a vital skill in demand by industry and somewhat at odds with what happened in the past, when they were motivated to acquire a useful skill that would be relevant in some future job or financially lucrative career. This may suggest that students are now approaching programming from an ill-informed position and they do not have a clear idea of the marketable skills they need to acquire. Their attitudes will obviously change as they go through their course. Some students may be discovering a genuine interest in programming or computing, if they do not drop out early.

In addition, the teaching (or more accurately learning) of programming is a problem in itself. Programming is a skill, a difficult skill to acquire. It is best learned by practice and, if students are to learn effectively, some at least of this practice will have to be self-directed. An instructor's key role is to persuade the students to do this and thus to motivate them. They must be motivated so that they will engage appropriately [3].

Unfortunately, motivation is an abstract concept that is difficult to measure in any meaningful way [3]. It is possible

¹ Maria Feldgen, Universidad de Buenos Aires, Facultad de Ingeniería, Argentina, mfeldgen@ieee.org

² Osvaldo Clúa, Universidad de Buenos Aires, Facultad de Ingeniería, Argentina, oclua@ieee.org

to observe a person's behavior and from that to infer their likely motivation, but it is never possible to be certain. Entwistle [4] describes three generic types of motivation:

- *extrinsic* - the desire to complete the course in order to attain some expected reward;
- *intrinsic* - deriving from an interest in the subject;
- *achievement - competitive* - based on "doing well" and (sometimes) better than peers.

During the first week, students answered surveys about their motivations on pursuing a Software Engineering degree and programming. The surveys showed that the motivations our students come with the first day they are sitting in our classroom are:

- He/she has no idea why he/she has to learn programming and he/she perceives it as a mandatory part of the chosen degree. He/she believes that people expect a student coming out of a programming course to be able to program any program, as well stated by David Gries[5]. No such expectations exist for a calculus or chemistry student, so he/she probably will "hate programming". This is a fact, the same as when he/she began high school, he/she must "hate Math" and any other subject that was defined as "very difficult".
- He/she is not sure what a software engineer does, but he/she believes that engineers build Web sites and games, because the name of the author is visible on the screen. Games are programs. This is not the fact with tradeoff products, as word processing, or banking system, these exist and you can buy them.
- He/she has a family commitment and, in many cases, is strongly influenced by parents or older brothers or sisters to pursuing this degree. This is a typical Hispanic culture commitment [6].

After these first week surveys we only qualify in the extrinsic type of motivation: they will complete the course as soon as possible determined to avoid any programming at no matter what cost.

Researchers are presenting a number of papers regarding techniques to improve the quality of instruction in the area of computer programming [7][8]. Much has been written about the most appropriate language and paradigm to use to teach such students, and the tradeoff between choosing a language for its pedagogical suitability or the extent of its use in industry. Since 1988, the authors have been struggling with many of the same problems and concerns discussed in the literature. We believe that a language or paradigm is only a tool for organizing our programs and our thoughts about them and that can be more or less pedagogically unsound. But it is not the key in motivating our students to learning programming. Students must engage by themselves in programming, as Felder[9] has said "people learn by doing, not by watching and listening". How do we promote an

intrinsic motivation, that our students will learn programming?

INTRINSIC MOTIVATION

Specific examples of teaching and learning strategies used in many institutions based on learning styles, showed that when students' individual learning style preferences are accommodated their motivation to learn increases. We focused on the Dunn and Dunn Learning Style Model and their five categories: Environmental, Emotional, Sociological, Physiological and Physiological/Cognitive Processing. Unfortunately, the institution does not authorize us to administer the PEPS (Productivity Environmental Preference Survey) [10] assessment instrument to our students.

In an institution with a tight time schedule and classroom assignments, we have no chance to make any environmental change, but we try to generate an informal environment in the classroom and labs. Programming courses can install and customize compilers and working environments in the labs PC's. To train our students in the use of the computer and working environment, they have as first lab assignment, the installation of a free Pascal compiler (Virtual Pascal) [11] and the program development environment. They learn to customize the environment, to change options, to watch the content of variables and debug a program setting breakpoints and executing step by step. They must do the same work at home, or in the other labs when they do their homework or out-of-class assignments. For freshman students this is valuable, because we trust them to maintain the lab.

For the other learning styles categories, we designed a set of tasks for in-class assignments, lab-assignments, homework and out-of-class assignments that appeal to all learning styles, as a way of promoting effective learning and encourage class participation. But these activities were more or less effective depending on the exercises we presented.

The exercises must catch students' attention or they remain undone. As freshman students they have limited abilities in dealing with written statements and asking questions. They must try to understand the problem statement. They must learn that programming is not only writing a lot of instructions in a computer language. It means first understanding the problem and then designing a solution that can be written using the known instruction set. To develop both their soft skills (reading, writing and communication skills) and programming skills we need their attention and participation.

Students favorite exercises context has changed in the last years, depending on what (computers, programs, etc.) had impressed and induced them to study computing. As students can not explain their desires, we have carefully analyzed how they did their homework. Homework has mandatory and elective exercises. If the problem statement context matches their preferences, they hand-in the whole program or an unfinished program with an explanation about their trouble with the language, for each exercise. There are no differences between mandatory and elective exercises. In the other case,

they argued that they did not understand the problem and asked for more time. Elective exercises are ignored.

Elective exercises show us how students' preferences change, and the impact that computing had on high school students. In figure 1, we show the percentage of elective exercises context students hand-in well solved. Contexts are traditional calculus and engineering exercises, business and games problem statements.

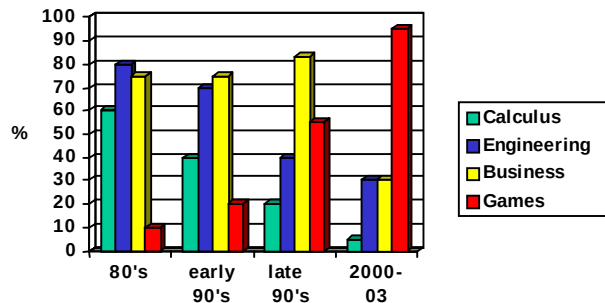


FIGURE 1
WELL SOLVED ELECTIVE EXERCISES BY CONTEXT

In the last years, the learning activities have been organized around the preferred context for our WEB age students: games. We introduce them to the other type of problems (or different contexts) but include them afterwards as part of a game, or support for the game, broadening students' view of computer applications.

Games and Design

Poor learning tendencies, as superficial attention, impulsive attention and staying stuck without adequate strategies are evidence of lack of reflective thinking. A way of avoiding these tendencies is to introduce every new concept trying that the students figure out why there is a need for the new concept and how it relates to external matters, i.e. what problems this new concept can solve that the old one failed on [12].

To support homework and out-of-class assignments we have in-class assignments. The objective of the tasks given to the students is that they require them to apply their knowledge to new situations (as reflective learners). Students need a referral to any earlier work that relates to it and to have the opportunity to extract their existing understanding to encourage them for discussions.

Students have different backgrounds, but each one, has sometimes played a game. Games have clear sequences of actions and strategies suited for procedural languages as PASCAL. All students understand a game problem statement because it is the way they expect the rules and strategies of a game. It is a common background or knowledge, a familiar situation, to begin with and to explore and develop students' problem-solving skills.

Problem solving exposes students to design tasks. Students must learn that they must focus first on understanding the problem. They must figure out how it works trying to write examples of input/outputs. Then, they must rewrite it in a sequence of actions, decisions and repetitions, and review if the specification works as expected using their examples. A closer look at the specification may require further specification of complex actions. Now they are ready to begin to code, compile, debug and deliver it. We use this approach for all exercises and assignments preventing students from trying to write code in PASCAL if they can not explain their solution in Spanish or making an example.

The first class before we introduce them to PASCAL, students learn how to write a program specification in structured Spanish with simple rules, as shown in table 1.

Write the sequence of actions needed to play a given game, using the following simple rules:

- for simple actions use short sentences beginning with a verb in infinitive (that represents the action)
- for decisions use: the word if, the condition for the decision, the actions if the condition is true, the word else and the actions if the condition is false, if any.
- for repetitive actions, use the word do, and how many times or until any condition occurs, write each of the repetitive actions and finish with end do.

Example:

Make a grid with 6 rows, a column for the round number from 1 to 6, and two columns more, one for each player.

Do until the two players have played:

- Do 6 rounds (or 6 times)
 - throw 3 dice
 - count the dices with the same number as the round
 - if the count is 3
 - write down 20 points on the grid (the column of the player and the row of the round).
 - else if the count is 2
 - write down 10 points on the grid
 - else if the count is 1
 - write down the round number on the grid
 - else write down 0 on the grid
- end do
- add the points in the column of the player
- end do
- if player 1 has more points than player 2
 - tell "player1 wins"
- else if player 2 has more points than player 1
 - tell "player2 wins"
- else tell "draw"

TABLE 1
RULES FOR PROGRAM SPECIFICATION

Students hate the specification writing task, but we have observed them, in the second semester, writing informal specifications for ordering their ideas or when they are discussing with others trying to find out the shorter solution to code.

Games and the Syllabus

Data types and variables are difficult programming language concepts. We introduce them to programming using constants, literals and only two data types: integer and boolean. Using dice, lottery and any other game with numbers, they learn first integer and boolean operations. In the in-class, labs and homework assignments, students design and program this type of games. The problem statement requires that students must verify all the rules of a game. The player can only win by one of the rules (the best), for all the other rules the program must display “false”, as shown in table 2.

- | |
|---|
| <p><i>A player throws 3 dice and wins with the best of the following rules (the program displays true for that rule and false for the others) or lose (the program displays false for all rules):</i></p> <ul style="list-style-type: none"> • wins 30 points when the player has 3 dice with the same value • wins 40 points when the values of the dice are different and consecutive • wins 20 points when the value of all the dice are odd • wins 10 points when the value of two of the dice are 2 and 3. |
|---|

TABLE 2
EXAMPLE OF AN IN-CLASS ASSIGNMENT

When students design, code and apply De Morgan’s Theorem well, we change the problem introducing strategies, where the player takes different actions depending on whether the rule is true or false, and ask “how much did the player win?” A program specification solving this kind of problems contains if sequences or nested ifs. It is time to introduce the Pascal IF and CASE sentences and work out the games with the new knowledge.

Then, we complete the game with more than two players, introducing control structures. After that, we introduce function and procedures and top down program structure. To continue working out the games, we introduce integer arrays, and then, all the other data types (character, enumerated, floating) because students ask for them. We finish the first semester with strings, multidimensional arrays and matrix processing algorithms.

The second semester, we introduce searching and sorting algorithms, files, pointers, recursion and stacks, queue and lists, using board-games and games which require concepts from Physics and Math as race games.

Homework includes different types of problems or contexts related to different applications. We reuse the same exercises in the next assignments, as the same problem grows and seems more and more a “complete” game or application. The goal is to review the essential points discussed in the past lectures, to prepare for the next topic and to introduce them to other application areas. Students can learn from their mistakes at a leisurely pace.

Homework assignment must be individually solved, following a strict presentation format and schedule. Any

homework assignment out of date implies handing-in all of the exercises, mandatory and elective, and a fine (more exercises). A fine is our negotiation for more time. Homework assignments have a review as feedback for the student, but not a grade.

The course has an out-of-class assignment (project) every semester. The first project has been, in the last two years, board games. We collected the rules and strategies from a Web site [13]. We made little changes in the rules and strategies to discourage our students from copying the solution from anywhere. They must design using function and procedures with multidimensional arrays and all the data types they have learned. The input/output of the game is not relevant. The second project is the same project but on the WEB, as CGI programs within an environment supporting players and scores. Input/output presentation and files are relevant.

GAMES AND LEARNING STYLES

To reinforce earlier material, we introduce some role-playing: students must play the role of designers and programmers. A designer writes the specification and “how it works” example about a game, which relates different problems solved before. Students interchange the specifications. A programmer writes the code for the specification, and returns both, specification and code, to the designer. This student verifies the program using his/her example. If the program does not work as expected, he/she must find out where in the specification the problem is, rewrite the parts and return it to the programmer. During these in-class assignments, students perceive the different learning styles of their classmates. A specification written by an analytic learner is difficult to understand for a global learner, or vice versa. Some characteristics of these kinds of learners as defined by Rita Dunn [14] are:

- An *analytic* learner - prefers to work alone, becomes deeply involved with the task at hand and processes it step by step, prefers one task at a time, remembers details, analyzes a problem first, doesn't like vague questions such as “suppose...”, is consistent with rules and assignments, prefers given options and wants specific constructive feedback.
- A *global* learner - needs to see the “big picture”, reads for overall idea, often skipping details, likes team competitions, lets others go first, likes to work with others, takes criticism very hard, sees relationships, understand things “in context”, can work on several projects simultaneously and relates what is taught to personal experiences.

Students with different learning styles approach problem solving in different ways, but games motivate both and complement them. Analytic learners help global learners with the analysis of the problem and the details of the rules. In their specifications every action is detailed, so that they make

a grouping of simple actions in complex actions in the last step, in a bottom up approach.

Global learners help analytic learners to find out quickly what kind of game it is. They build the “how it works” example using different strategies together with the specification. The specification is written in a top down approach with less detail. Global learners hate the last step of writing the specification when they must refine complex actions. They need our feedback to know they are doing their work well, they discuss every step with us, while they are imagining the different game situations, contributing with new details for the analytic learners.

We observed that when students have the same learning style (designer and programmer), they are able to obtain a program from a brain dead specification. It is an incomplete or chaotic game. If they have different learning styles with a useless specification, they cannot code the program. But the final program of mixed learning style students has a better structure than that of the same learning style students. In addition, these tasks help them to develop their soft skills and to socialize. This is one of the most motivating tasks for the students and they are proud of working as “real” software engineers.

CONCLUDING REMARKS

We have the highest rate of attendance in class according to the last three years of University statistics. 60% of the students who finished the course passed the examinations with high marks before beginning the next course and 95% did it after one semester. Before we introduced the game approach, 25% of the students' left the course and 10% deserted from the Institution. In the last two years we observed a 10% drop out and no desertion; this 10% takes the course again in the next term [15].

In table 3 we show students' behavior and performance in the last four years, and how the Math / Physics course requirements impacted our course. Math and Physics courses

are prerequisite for courses other than Introductory Programming. When students fail in the mid-term exams of Math and/or Physics, they drop out our course, even if they have passed our course mid-term exams. In addition, Math and Physics courses introduced remedial courses and students were informed in 2002 after the first semester. In 2003, tutors informed the students at the beginning of the year. This year, we have observed that they were more relaxed about their performance in the other courses, until the second semester when they began to fail. When they began with the remedial courses, they had no time for our subject and the number of students that could not finish the second out-of-class assignment increased.

The column "Switch to other Institution" in the table shows the effect of financial hardship of our students' parents. This survey is made in an institution with an expensive tuition fee and a country with an increasing unemployment rate. Students drop out even if they have good marks. In Argentina, they have a chance to enter a National University with no tuition fees, but a lot more demanding in performance.

Students' surveys revealed that our course was key for them to remain in the institution to finish their studies, even if they fail in any subject, as shown in table 4.

The traditional algorithms, control structures, data types and related concepts are abstract and difficult to explore, and they do not appear as stimulating issues by themselves for our students. We introduced what students regard as their real world: games and Web-programming. This approach helped us to promote an effective learning environment and encouraged class participation.

Programming activities are well suited to accommodate all learning styles and make teaching and learning a fun activity, helping them to make the transition from high school to college.

TABLE 3
STUDENTS BEHAVIOR AND PERFORMANCE

Year	Gender	Number of Students	Math / Physics drop-out	%	Fail	%	Switch to other Institution	%	Pass	%
2003	Male	49	12	24.49 %	9	18.37 %	4	8.16%	26	53.06%
	Female	8	1	12.50 %	4	50.00%	0	0 %	3	37,50%
2002	Male	57	12	21.05 %	5	8.77 %	5	8.77 %	40	70.18 %
	Female	9	3	33.33 %	1	11.11 %	2	22.22 %	5	55.56 %
2001	Male	68	12	17.65%	9	13.24 %	8	11.76 %	47	69.12 %
	Female	10	3	30.00 %	2	20.00 %	0	0.00 %	5	50.00 %
2000	Male	53	12	22,64 %	6	11.32%	0	0.00 %	35	66.04 %
	Female	11	4	36.36 %	1	9.09 %	0	0.00%	6	54.55 %
1999	Male	49	10	20.41 %	6	12.24%	0	0.00 %	33	67.35 %
	Female	14	3	21.43 %	4	28,57 %	0	0.00%	7	50.00 %

TABLE 4
STUDENTS WHO TAKE THE COURSE AGAIN

Year	Gender	Number of students	Year - 1					Year - 2
			%	Math / Physics drop-out	Fail	Drop out from Institution	%	Fail final exams
2003	Male	19	100.00 %	12	5	0	0.00 %	2
	Female	4	100.00 %	3	1	0	0.00 %	0
2002	Male	23	100.00 %	12	9	0	0.00 %	2
	Female	5	100.00 %	3	2	0	0.00 %	0
2001	Male	19	88.89%	12	6	2	3.77 %	2
	Female	4	80.00 %	4	1	1	9.09 %	1
2000	Male	10	62.50 %	10	6	6	12.24 %	0
	Female	4	57.14 %	3	4	4	28.57 %	1
1999	Male	12	75.00 %	10	4	6	12.24 %	4
	Female	5	71.43 %	4	0	2	28.57 %	1

Students made an oral presentation of their second out-of-class assignment to the dean, other professors, relatives and friends. The presentation was not mandatory. Later, the Web site administrator informed us, that students had access their WEB CGI programs from outside the University. Some of them told us that they showed their work to their friends and relatives during their vacations. They suggested us that we should show the projects to the freshman students entering our course this year. They committed themselves making the presentation as a way of telling freshman students that he/she can survive the first year at University and have fun learning programming.

REFERENCES

- [1] Nelson, B, Dunn, R. Griggs, S. A., Primavera, L., Fitzpatrick, M. et al, "Effects of Learning Style Intervention on College Students' Retention and Achievement", *Journal of College Student Development*, Vol. 34, September 1993, pp.364-369.
- [2] Hein, T. L. , Budny, D. D, "Learning Styles, "Applications in the Physics and Engineering Classroom (Invited paper)", Proceedings ICECE'99, Rio de Janeiro, Brasil. August 1999, pp 492-497.
- [3] Jenkins, T., "The Motivation of Students of Programming" *SIGCSE Bulletin*, Vol. 33 No. 3, September 2001, pp 53-56.
- [4] Entwisle, N., "Motivation and Approaches to Learning: Motivating and Conceptions of Teaching", *Motivating Students*, Kogan Page, 1998.
- [5] Gries, D, "Where is Programming Methodology These Days?", *SIGCSE Bulletin*, Vol. 34 No. 4, December 2002, pp 5-7.
- [6] Dunn, R., "Hispanic American Students and Learning Style", *Emergency Librarian*, Vol. 23 No. 2, November-December 1995, pp. 11-16.
- [7] Budny, D. , Luna, L., Viperman, J., Patzer, J.L., "Four Steps to Teaching C Programming", Proceedings 32nd ASEE/IEEE Frontiers in Education Conference, Boston, MA, F1G-18-F1G-22, November 2002.
- [8] Cordes, D. , Parrish, A., "Active Learning in Computer Science: Impacting Students Behavior", Proceedings 32nd ASEE/IEEE Frontiers in Education Conference, Boston, MA, T2A-1- T2A-5, November 2002.
- [9] Felder, R.M., "Beating the Numbers Game: Effective Teaching in Large Classes", 1997 ASEE Annual Conference, Milwaukee, WI, 1997.
- [10] Dunn, R. , Dunn, K., Price, B. , *Productivity Environmental Preference Survey: An Inventory for the Identification of Individual Adult Preferences in a Working or Learning Environment*, 1990, Price Systems Inc. Lawrence, KS, <http://www.learningstyle.com>
- [11] <http://www.vpascal.com>
- [12] Carbone, A., Hurst, J., Mitchell, I., Gunstone, D., "Characteristics of Programming Exercises that lead to Poor Learning Tendencies. Part II", *SIGCSE Bulletin*, Vol.33 No. 3, September 2001, pp. 93-96.
- [13] <http://www.acanomas.com.ar/reglamentos>
- [14] Dunn, R. , Beaudry, S., Klavas, A., "Survey of Research on Learning Styles", *Educational Leadership*, Vol. 46 No. 6, March 1989, pp 50-58.
- [15] Feldgen, M. , Clúa, O., " New Motivation are required for Freshman Introductory Programming (Work in Progress)", Proceedings 33rd ASEE/IEEE Frontiers in Education Conference, Boulder, CO, T3C-24, November 2003