USING BOARD GAMES OBJECT-ORIENTED PROGRAMMING FOR TEACHING MATHEMATICS

Alexander Vaninsky,
Hostos Community College, CUNY
500 Grand Concourse, Room B 409, Bronx, NY 10451, USA.
email: avaninsky@hostos.cuny.edu

Introduction. Successful teaching and learning mathematics comprises two indistinguishable components: mathematics relation to the real world and mastering internal logic of mathematics as a scientific discipline. Importance of the first component is out of question; for instance, Principles and standards for school mathematics (2000) stress the development of applications skills in several rubrics. It is less mentioned that understanding mathematics *per se*, as an abstract science, is also of crucial importance, because it underlies mathematics applications skills.

Contemporary school and college programs are designed so that most of students, for example, remain unaware of the nature of real and complex numbers, do not perceive vectors and polynomials adequately, and have no or very little knowledge about mathematical proofs. This situation restricts their capability to recognize same mathematical categories presented in different forms and surf easily among them. Thus, students could not fully grasp and use an idea that vectors in plane, complex numbers and fractions are just ordered pairs of real or, correspondingly, integer numbers equipped with different operations of addition and multiplication. It should be noted that general understanding of the nature of abstract mathematical objects is practical, because it forms a basis for better learning of more comprehensive topics in mathematics.

Seeking a way of teaching abstract mathematical topics using "tangible" tools author have found that object-oriented programming of board games is a convenient one. This paper presents some examples and demonstrates, in particular, independence of mathematical reasoning from the nature of objects and shows how mathematics can be applied to decision-making. Suggested approach teaches students that mathematics may be considered as a science about properties of abstract objects rather than about objects themselves. It helps development of critical thinking capabilities so needed in studying science and engineering.

Object-oriented programming and mathematical systems. Object-oriented programming operates with classes and objects that possess properties and are subject to operations called methods. A class defines abstract characteristics of an entity in question, while an object represents a particular instance of a class. Three main features of classes are essential for this paper: polymorphism, encapsulation, and inheritance. For objectives of this paper, polymorphism may be considered as an abstraction from any actual class implementations, encapsulation, as closure with regards to methods, and

inheritance, as a possibility to extend a class to a wider one with more properties and methods, using the given class as a building block.

A mathematical system contains a set of concepts and elements, a set of axioms related to the concepts and elements, and a set of operations over the elements of the system. Straightforward analogy between mathematical systems and classes may be set up as follows. Class may be considered as a mathematical system, objects of a class, as a set of its elements, properties, as axioms, and methods, as applicable operations. To make students familiar with the suggested approach, it is practical to begin with consideration of natural numbers as a class, to point out its properties and methods, and then to expand it to the classes of whole, integer, rational, and real numbers.

Object-oriented approach to problem solving. A problem related to problem solving, borrowed from Dressler &Keenan(1998), is used as a basic example. It is considered as a board game and extended further to more general object-oriented setting. It is used in this section to demonstrate how object-oriented programming approach may be used for solution of similar problems. It is also shown that suggested approach provides an opportunity to see a bigger picture rather than allows just solving particular problem.

The problem is this. "Three friends, Jane, Rose and Phyllis, study different languages and have different career goals. One wants to be an artist, one a doctor, and the third a lower. <The following rules determine their choices:> (1) The girl who studies Italian does not plan to be a lawyer.(2) Jane studies French and does not plan to be an artist. (3) The girl who studies Spanish plans to be a doctor. Phyllis does not study Italian. Find the language and career goal of each girl."

To apply object-oriented programming, we first introduce a class **Student** with two properties: ***Language*** and ***Career***. Property-1 ***Language*** has values {*French, Italian, Spanish*}, property-2 ***Career***, values {*Artist, Doctor, Lawyer*}. The class has three installments, or objects, in this problem that may be labeled by the first letters of the girls' names: **J**, **R**, and **P**, respectably. A method called **MatchGoals()** sets up correspondence among the properties of the objects. It works with a matrix dimension 3 by 3 with rows corresponding to the values of property-1 (***Language***), and columns, to property-2 (***Career***). An algorithm of the method processes rules successfully and fills in cells of the matrix with the letters corresponding to the objects that can occupy the cell, **J**, **P,** or **R** or their logical combinations using ~(NOT), $\wedge$(AND) and $\vee$ (OR) symbols. Eventually, only three cells are filled in, each taken from one row and one column; all the rest cells are empty.

The algorithm performs several runs using the rules consequently, and analyzes the table after each run. Run 1 uses Rule (1) and empties a cell locating at the intersection of row *Italian* and column *Lawyer*, see Fig. 1, Run 1, where symbol "-" stands for "**Empty**" that means that a cell contains no object name and should not be analyzed in consecutive runs. Run 2 uses Rule (2) and fills in row *French* with **J** and column *Artist* with **~J**, see Fig. 1, Run 2. After that the algorithm performs analysis of the table and changes cell *French-*

Artist for **Empty**, because it contains a contradictory $\mathbf{J} \wedge \sim\mathbf{J}$, see Fig. 1, Analysis 1. Run 3 utilizes Rule (3), and labels cell *Spanish-Doctor* as chosen one, see Fig. 1, Run 3. A value of this cell is not known at this stage but it allows marking all the rest cells in row *Spanish* and column *Doctor* as **Empty**, see Fig. 1, Analysis 2a. Now, the choice of cells is complete because each row and column contain only one non-**Empty** cell, see Fig. 1, Analysis 2b. Content of the cell *French-Lawyer* is **J**. Run 4 fills in a cell *Italian-Artist* with $\sim\mathbf{P}$, using the Rule (4), see Fig. 1, Run 4. After that Analysis 3 step changes $\sim\mathbf{J} \wedge \sim\mathbf{P}$ for **R**, see Fig. 1, Analysis 3a, and then finalizes the process by matching remaining object **P** with the cell *Spanish-Doctor*, see Fig. 1, Analysis 3b.

Figure 1. Steps of the **MatchGoals()** method.

Run 1. Using Rule (1).

	Artist	*Doctor*	*Lawyer*
French			
Italian			-
Spanish			

Run 2. Using Rule (2).

	Artist	*Doctor*	*Lawyer*
French	$\mathbf{J} \wedge \sim\mathbf{J}$	**J**	**J**
Italian	$\sim\mathbf{J}$		-
Spanish	$\sim\mathbf{J}$		

Analysis 1. Logical analysis and transformations.

	Artist	*Doctor*	*Lawyer*
French	-	**J**	**J**
Italian	$\sim\mathbf{J}$		-
Spanish	$\sim\mathbf{J}$		

Run 3. Using Rule (3).

	Artist	*Doctor*	*Lawyer*
French	-	**J**	**J**
Italian	$\sim\mathbf{J}$		-
Spanish	$\sim\mathbf{J}$		

Analysis 2a. Logical analysis and transformations.

	Artist	*Doctor*	*Lawyer*
French	-	-	**J**
Italian	$\sim\mathbf{J}$	-	-
Spanish	-		-

Analysis 2b. Logical analysis and transformations.

	Artist	*Doctor*	*Lawyer*
French	-	-	**J**
Italian	$\sim\mathbf{J}$	-	-
Spanish	-		-

Run 4. Using Rule (4).

	Artist	*Doctor*	*Lawyer*
French	-	-	**J**
Italian	**~J ∧ ~P**	-	-
Spanish	-		-

Analysis 3a. Logical analysis and transformations (continued.)

	Artist	*Doctor*	*Lawyer*
French	-	-	**J**
Italian	**R**	-	-
Spanish	-		-

Analysis 3b. Logical analysis and transformations Final result.

	Artist	*Doctor*	*Lawyer*
French	-	-	**J**
Italian	**R**	-	-
Spanish	-	**P**	-

Object-oriented approach allows easy generalization to numerous similar applications. Thus, I suggested reconsidering this problem using class **Nature** that inherits all properties and methods of the previous class **Student**. Property-1 was renamed as ***NumberOfLegs*** with the set of values $\{0,2,4\}$; property-2, as ***TypeOfMoving*** with values {*Flying, Walking, Swimming*}. The objects of the new class were **Animal**, **Bird**, and **Fish**. Students were asked to solve the new problem using the same method and to prepare other examples of the same class with different objects and different interpretations of its properties. While discussing suggested interpretations, author stressed that rules should be chosen very carefully. Thus, there cannot be too little rules, because its number should be enough to fill in all cells. On the other hand, there should not be too many rules, because in this case the rules may be either redundant or contradictory. It may be demonstrated also that different collection of rules may lead to the same result, thus being equivalent. For example, the Rules (1) through (4) above may be changed for the following equivalent set: (1a) Jane studies French and plans to be a lawyer. (2a) Rose does not study Spanish. (3a) Phyllis plans to be a doctor.

Students were asked to solve the same problem with a different set of rules and to show that answer is the same. In the following discussion, students were asked, whether the subject area (career choice, skills etc.), type of the participants in the problem (human beings, animals etc.), or the character of properties in question (career choice, preferences, tastes etc.) were principal for the strategy of solution. While arriving at the answer "No", they were asked to design a class that covers all such problems, that makes a link to the subject area only when objects are instantiated. Table 1 presents such class. It contains: (i) Two properties Property-1 and Property-2, respectively, each having a set of values, that should be matched; (ii) A new Conclusion-property ObtainedResult that has three possible values: OK, Redundancy, and Contradiction. This property indicates whether a problem in question was solved (OK), contains the rules that are not needed for solution (Redundancy), or cannot be solved due to contradictory rules (Contradiction);

(iii) Class-related rule-properties that do not refer to an object name, like Rules (1) and (3) above; (iv) Object-related rule-properties that use an object name, like Rules (2) and (4) above; (v) Class method **MatchGoals()**. The class, by consensus, was given a name **Matching**, see Table 1. Students were also suggested to reconsider the problem, using object **Nature** of class **Matching** that actually inherits all properties and methods of the class **Student**. Property-1 was renamed as ***NumberOfLegs*** with the set of values $\{0,2,4\}$; property-2, as ***TypeOfMoving*** with values $\{Flying, Walking, Swimming\}$. The objects of the new class were **Animal**, **Bird**, and **Fish**. Students were asked to develop a new wording for the new problem and to solve it using the same method. After that they were asked to think off a series of board games based on class **Matching** as different objects with specific properties.

Table 1. Abstract class **Matching**.

Properties	
Property-1	Values: $A,B,C,...$
Property-2	Values: $a,b,c,...$
ObtainedResult-property	Values: $\{OK, Redundancy, Contradiction\}$
Class-related rule-properties (CRPs) Properties expressed in terms of the Property-1 and Property-2 values.	
CRP-1	Examle: $\{A \wedge c = True\}$
CRP-2	
....	
CRP-m	
Object-related rule-properties (ORPs) Properties expressed in terms of the Property-1 and Property-2 values and including the name of an object.	
ORP-1	Example: $\{B \vee a = True$ for this object$\}$
ORP-2	
....	
ORP-n	
Methods	
Class-related method **MatchGoals()**	Algorithm of the method.

Binary system and logical operations in Nim game programming is another example of using board games programming for teaching mathematics. It focuses on mathematical tools that might be needed for implementation of the methods. Detailed description is given in Vaninsky (2007).

References.

Dressler, I. and E. Keenan. (1998). *Integrated mathematics: Course 1*. 3rd Ed. NY: Amsco School Publications.

Vaninsky, A. (2007). Activity - based introduction to the binary system: Nim game winning strategy. *International Journal of Mathematical Education in Science and Technology,* January 2007, 38(1), 43 - 54.