AlgoPath's New Interface Helps You Find Your Way Through Common Algorithmic Mistakes

Estelle Perrin
CReSTIC
IFTS - University of Reims Champagne-Ardenne
Charleville-Mézières, France
estelle.perrin@univ-reims.fr

Sébastien Linck
IFTS - University of Reims Champagne-Ardenne
Charleville-Mézières, France
sebastien.linck@univ-reims.fr

***Abstract*—This paper presents the new interface of our serious game AlgoPath and its related interactions. AlgoPath helps students learn algorithmic. The virtual world represented in AlgoPath is all linked to the business of road construction and people running along these roads: objects students interact with are 3D figures, houses (huts and suburban houses), boxes, a crane, a concrete mixer and a bus station. This paper shows that AlgoPath helps students avoid common mistakes they can make while learning algorithmic. The entire interface is dedicated to help them conceptualize and understand the rules of algorithmic and programming. Whenever it is possible, AlgoPath reminds students of these rules and corrects the mistakes.**

***Keywords*-3D-based training; education; algorithmic; ludic teaching**

I. INTRODUCTION

When we first presented AlgoPath last year [1], we introduced the reasons why we had wanted it to be implemented: it was a necessity to have an entertainment computer program in which students could learn algorithmic but every single computer program we had looked into was simply an improved imitation of flowcharts. To renew interest, motivation, and enjoyment while learning, we had to achieve a virtual world in which students could create any algorithm they wanted. But to resemble video games, it had to be a world so we had spent time thinking of what a good concept of an algorithm could be. We had focused on how a variable should look like and had decided to turn it into a 3D white figure carrying a backpack that contains a value. In AlgoPath, a 3D white figure runs on a stone path. A path shape is related to the algorithmic statement it represents: linear when it is an assignment, forked when it is a conditional statement and circular when it is a loop.

During this year, we carried out a survey to show if students liked or disliked AlgoPath. Fifty students were quizzed. They came from different courses of studies - a half from under-graduate courses dedicated to websites; the other from scientific general under-graduate courses - but all had to attend a course dedicated to algorithmic. One hundred percent of students said they were thrilled to learn algorithmic with a game but sixty-five percent said the interface (see Figure 1) was neither ergonomic nor ludic.

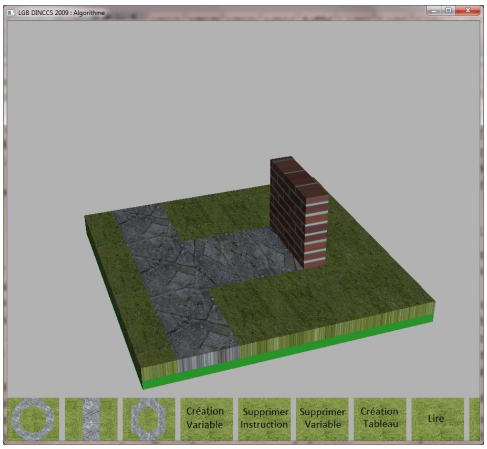

Figure 1. AlgoPath's old interface. Only the big buttons at the bottom can be selected.

They were right because AlgoPath was only a prototype implemented to validate the concepts, and we had not focused on the interface as we should have. So, the interface objects were succinct: ten big buttons displayed at the bottom of the window to create the basic statements of algorithms. Furthermore, to see the body statements of a loop or a conditional, students had to click on the corresponding bush along the stone path. Then the window completely changed and only showed statements of the loop or the conditional. Because the interface acted in that way, the students could no longer keep the whole concept in mind and were confused.

In this paper, we present the new interface of AlgoPath. This interface was designed (1) to avoid common mistakes students can do while learning how to create algorithms for the first time and (2) to navigate through a one and only world. The related works (see section II) show that a serious game interface must be close to a video game interface to be efficient. They also show that serious games can help to learn methods and rules proposed by a class. Section III is

Copyright (c) IARIA, 2013. ISBN: 978-1-61208-250-9

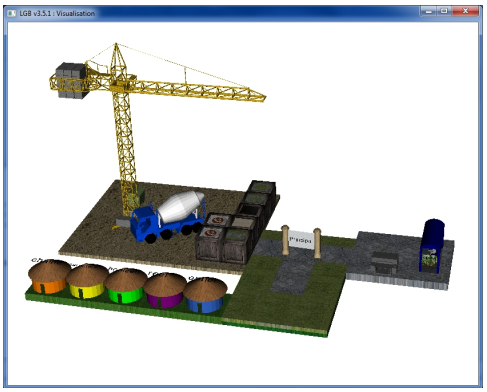

Figure 2. AlgoPath's new interface.

dedicated to the new interface of AlgoPath and the common mistakes it avoids. Section IV concludes the paper and shows how the evolution of AlgoPath might be.

II. Related Works

We can find many works on serious games. We will focus on three interesting characteristics of this kind of games: the creativity developed by the gamer, the graphical interface and environment of the game, and the usefulness of it.

A. Creativity

Serious games are made to learn something. Students can either acquire knowledge about courses studied at school or validate this knowledge with serious games.

Some are just made up of questions about the subject taught at school. A Quiz [2] or a maze with a range of issues related to a specific topic, [3] or [4], do not bring any part of creativity in the game. This first type of serious game has no creative part, it is more or less like school exams. Some of these games are labelled "serious games" only because of their simplified interface and their childish presentation, although it is no real support for learning whatsoever.

But on the other hand, some games develop a kind of interaction between the learner and the game, where the learner is involved in creating the game: for example series of actions by the player to create a solution to advance the storyline. The game does not only serve to check the acquisition of knowledge but actually allows the student to learn the concepts through play like: see [5] in physics, see [6] to learn problem-solving or see [7] in mechanical engineering. In [5] the authors show that the creativity increases when the player can share his creation with other players .

B. Interface

Serious games are first of all games, and so visual representation and interactions between the player and the interface of the game are very important. So, for people who play serious games, a game must be attractive in its form and in its way of learning. Today all the most played video games work in a 3D environment. If a serious game wants to be interesting for children or students, it has to be close to a classic game visual.

Questioning games are classically in only two dimensions. But some of them use 3D interface to interact with the player. These 3D-games are often made in 3D because of the subject of the game: geography [8] or architecture [9], but some are just made in 3D to have a better interface for collaboration between players [10]. Some serious games are just serious scenarii based on a commercially available video games [11], [12]. For example in [12], they use Tycoon City: New York® and SimCity Societies® games to learn daily economics and global issues.

But all these 3D interfaces and interactions do not frame the learning.

C. Usefulness

Many papers have been written about the utility of games in education. Whatever the age of the students, the learning with serious game is equivalent to that done in a master course[13]. For [14], serious games are no obstacle to students' success. From primary school to high school, the enjoyment to play [15] and, so to learn, is a good indicator of the usefulness of serious games.

To be effective, a serious game must be a game as well as a teaching aid. In that respect, teachers must be given a special training to be able to use serious games properly[16]. Then, if both teachers and learners use the game in the right way, then the game itself will gain in efficiency as well as in usefulness.

We have presented many serious games, some of which allow the students to get interested in discovering ans learning new subjects. We will next show how the new interface of AlgoPath promotes the learning of algorithmic.

III. AlgoPath's new interface

AlgoPath's interface was totally rethought in order to simplify the navigation and to help students avoid most of the common mistakes they can make while they learn algorithmic. The world within witch students can play is divided into five zones. The first one is the stone path. This zone represents the algorithm being built and it evolves gradually when students interact with the environment. The second one is a group of five houses. While interacting in this zone, students can create variables. AlgoPath provides five basic data types: integers, floats, Booleans, strings and characters. The third zone is the construction zone. Students can create aggregate or composite data type, used to represent entities that are described by multiple attributes of potentially different types. Students can create statements and they can build the prototype of a function or a procedure. The fourth zone is the bus station. This zone is dedicated

Copyright (c) IARIA, 2013. ISBN: 978-1-61208-250-9

to the simulation of the execution of the algorithm. And the fifth zone is the variables area. The latter is missing in Figure 2. because it is related to the memory usage and at the beginning there is no variable declared.

In the next sections, we describe the last four areas in details. The first one (the algorithm zone) was fully described in our previous paper [1] but, as the bush statements world totally replaced the main statements world when the user wanted to focus on the bush statements, we include a section in this paper in which we explain how we changed that.

A. Algorithm Zone

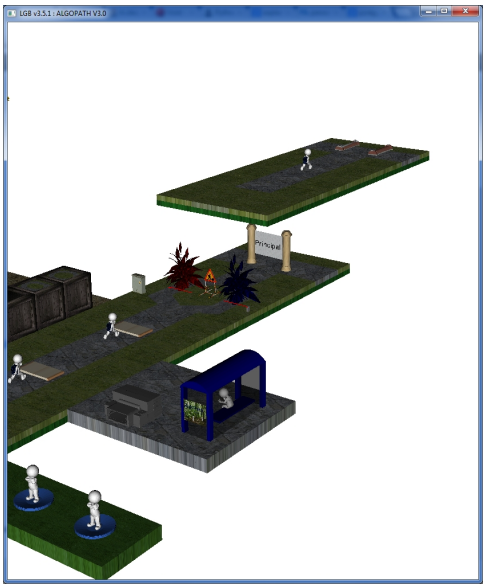

Figure 3. Level 0 and level 1 open.

In AlgoPath, a bush is a sequence of statements that describes actions to be performed. Replacing a world with another when students clicked on a bush to see its relative statements was not satisfying. Students could not locate themselves in the levels of decomposition of the algorithm. In this new version of AlgoPath, whenever students want to focus on statements of an else-part, a then-part, a loop, a function or a procedure, AlgoPath adds a new floor above the mother statement (see Figure 3). In that way, students do keep in mind the all concept. But, since a conditional statement can lead to two sequences of statements of the same level, AlgoPath does not allow to open more than one floor at a given level. Therefore, each opened floor of a superior level has to be closed before another floor can be opened.

B. Urban Zone

In the urban zone (see Figure 4) AlgoPath provides five basic data types: integers, floats, Booleans, strings and characters. Each data type has its own hut. The population

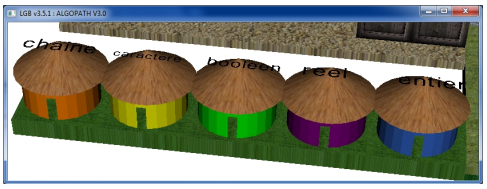

Figure 4. Urban zone.

of integer variables lives in the integer hut. The name of the data type is on the top of the hut and can be easily seen by students. Students can click on a hut as they would knock at a door and then trigger a variable. Clicking on a hut is one of the only few things students can do at the beginning of a session with AlgoPath (apart from creating a new composite data type; creating the prototype of a function or a procedure; and adding an output statement).

AlgoPath requires a name to declare a variable. If the name is the same as the one of an existing variable, AlgoPath declines the declaration and alerts students a variable has already been declared with this name. Just like in programming, words separated by a space are not accepted.

C. Memory Zone

The memory zone shows each declared variable. As seen in [1], a standing 3D figure personifies a variable. In this new version it stands on a pedestal whose colour is the same as the roof of the hut the 3D figure belongs to. A 3D red figure means the variable has not been assigned a value yet.

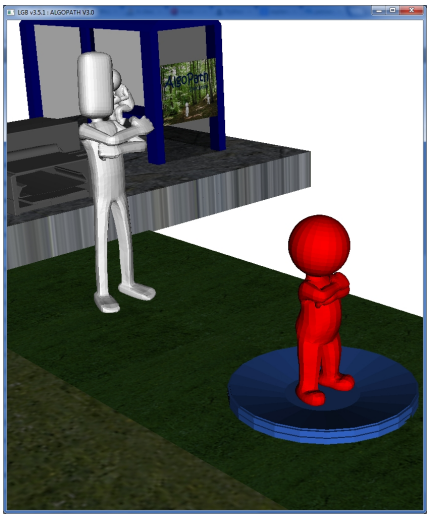

Figure 5. Two kinds of 3D figures.

Two kinds of 3D figures can stand on the memory zone (see Figure 5). A small and a little overweight one represents a basic data type variable; on the contrary, a tall and thin one represents an aggregate or composite data type variable. If students click on a tall and thin 3D figure, it opens a floor above it, with the 3D figures corresponding to the members

Copyright (c) IARIA, 2013. ISBN: 978-1-61208-250-9

of the composite or aggregate type, just as a new floor opens when students click on a bush.

D. Construction Zone

In the construction zone, students can find boxes (to create statements), a concrete mixer (to create new prototypes of functions or procedures), a crane (to create an aggregate data type), and a dustbin (to delete things). Each object is dedicated to an interaction described in the following sections.

1) Concrete mixer: If students click on the concrete mixer, Algopath helps them go through the creation of the prototype of a function or a procedure. When writing algorithm on a sheet of paper, students often forget syntax or are embarrassed deciding if they have to write a procedure or a function. According to our rules, if a module has only arguments by value and one result then it is a function. In any other case, we ask students to write a procedure (meaning a mix of arguments by value and by reference with zero or at least two results). First AlgoPath asks them if it is a function or a procedure that will be created. The name of the module is then required and AlgoPath launches the process of creating arguments. An argument is totally defined by its name and its type. AlgoPath lets students choosing the name but it automatically suggests the set of the available data types. So, students cannot specify a data type they have not already defined. But when you define a procedure in a programming language, you have two choices regarding how arguments are passed to it: by reference or by value. In AlgoPath, you have three choices: by input, by output, and by input-output. Passing by input refers to a way of passing arguments where the value of an argument in the calling function cannot be modified in the called function. Passing by output refers to a way where an argument has no value in the calling function prior to the called function, but the called function has to assign it a value. Passing by input-output refers to a way of passing arguments where the value of an argument in the calling function can be modified in the called function. There is a distinction between argument by input-output and argument by output because we want students to be aware that an argument by reference may not be assigned a value when it reaches the called procedure. If students want to create a procedure with zero or several arguments by input and only one argument by output then AlgoPath warns them that it should be a function instead of a procedure and adds it as a function.

Once the prototype of the module is defined, a new box appears in the construction zone. This new box is selectable. If students select it then a new floor opens in AlgoPath. It shows the - empty - body of the module. Next to the stone path where statements will be added, there is a zone that looks like the memory zone of the main algorithm. Instead of showing the main variables, it shows the arguments of the module. Visible features (see Figure 6) help the students

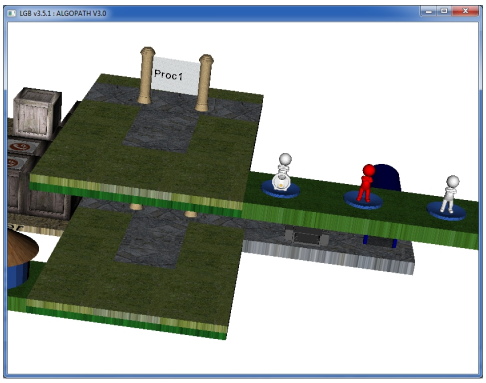

Figure 6. Three kinds of arguments (from left to right: by input, by output and by input-output).

recognize if it is an argument by input, by output or by input-output. An argument by input is a 3D white figure in front of which is a padlock. It means this variable was assigned a value and this value cannot change. An argument by output is a 3D red figure. It means it has not been assigned a value yet. An argument by input-output is a 3D white figure without a padlock. It means it has already been assigned a value and this value can change.

When students add an assignment statement in the body of a procedure, AlgoPath suggests them the set of variables available. Arguments by input are not included in this set. It is a little bit restrictive regarding programming, but it helps them understand that an argument by input cannot change its value.

2) Crane: Clicking on the crane launches the creation of composite data types. Students can create arrays or structures. An array is a set of consecutive variables of a same type. A structure is a collection of one or more variables, possibly of different types, grouped together under a single name for convenient handling. The variables named in a structure are usually called members. Students have to specify the types of cells of an array and the members of a structure. AlgoPath helps them with this process by suggesting the data types available. That is how AlgoPath avoids common typing mistakes made by students - writing a data type that does not exist or wrongfully writing a data type that actually exists. But it also helps the students understand that the most inner data types must be created first. If a person structure has to be created - which we will assume has a name and a date of birth - the date structure has to be created first. Naming an array, a structure and its members is another process students have to go through. AlgoPath verifies the integrity of the names. Naturally, a structure member and an ordinary (i.e., non-member) variable can have the same name without conflict, since they can always be distinguished by context.

3) Boxes: The construction zone contains seven boxes. Each box creates a new statement in the path of AlgoPath.

Copyright (c) IARIA, 2013. ISBN: 978-1-61208-250-9

Figure 7. The seven boxes to create statements.

Figure 7 shows the seven boxes. The lower row contains - from left to right - a box to create an assignment, a box to create an input statement, a box to create an output statement, a box to create a conditional statement and a box to create a loop. Since these five statements were fully described in [1], this section will focus on the last two boxes: the one with a receiver and a ”F” and the one with a receiver and a ”P”. But let’s just add that anytime the name of a variable is required, AlgoPath suggests a set of proper names available. To fully explain AlgoPath does not only check the names of the variables declared, it also computes the names of the variables of basic data types because one of our rules is that a variable of aggregate data type cannot be assigned a value. For example, if the variable P is declared as a person, that is a structure with two members - a name and a date of birth - whose names are ”name” and ”date” and, if a date is a structure with three members named month, day and year then AlgoPath suggests the following set of names: P.name, P.date.month, P.date.day and P.date.year. P and P.date do not belong to this set because they are composite data type variables.

The upper row left box creates a function calling statement, while the right box creates a procedure calling statement. These creations can occur within two contexts. The prototype of the function or the procedure may be already defined or not. If it is, AlgoPath helps students by notifying the arguments of the module and their status: by input, by output and by input-output. For each, it reminds students if they can associate a value or a variable of the calling module. Naturally, AlgoPath acts differently whether an argument of the module is exclusively by input or not. If it is, AlgoPath lets students choose if they will associate a value - an expression - or a variable. If it is not, AlgoPath only shows variables of the same data type of the calling module. If there is none, AlgoPath cancels the process of creating a new statement and explains why. Then students learn they first have to create variables in the calling module if they want to put results in them. If the prototype is not defined, AlgoPath asks questions so the students are able to define the calling arguments of the module. First, it asks what the name of the module is. Then it asks if students want to add an argument. If they do, it wants to know if the argument is a value or a variable. In case it is a value, the argument is automatically defined by input. In case it is a variable, students can choose if it is by input, by output or by input-output. At the end of the process, the prototype of the module is automatically created and a new box is added in the construction zone.

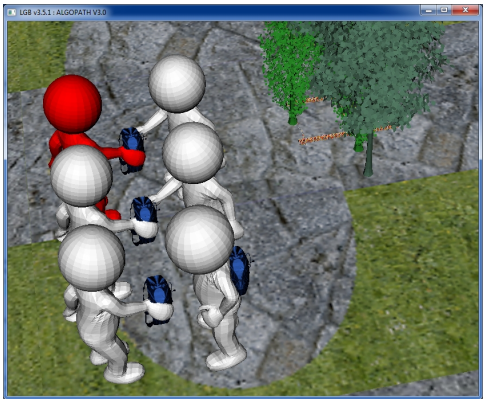

Figure 8. Visible features of calling arguments (colors and backpacks).

As mentioned in [1], there are visible features to tell the arguments apart. But in this version of AlgoPath, a calling argument by output is red while a calling argument by input-output is white (see Figure 8).

4) Dustbin: In the construction zone, there is a large commercial refuse bin. It opens when students click on it if a statement has already been added. Otherwise it does nothing. It closes when students choose the statement they want to delete.

E. Bus Station Zone

The bus station zone is under construction so we won’t talk much about it. This zone is dedicated to the execution of an algorithm. We expect a lot from this zone because normally students have to wait for the implementation of algorithms in a chosen programming language to discover the execution. Let’s just say a bus will drive along the path of AlgoPath to pick up the 3D figures (see Figure 9).

Figure 9. Execution.

Copyright (c) IARIA, 2013. ISBN: 978-1-61208-250-9

The interface will act accordingly: for example, gates will open or close, 3D figures will tell the content of their backpacks, etc. Moreover, when students click on the printer, AlgoPath shows them the algorithm writing with conventional declarations and statements.

IV. Conclusion and Future Work

[1] presented the different 3D objects to represent the concepts of algorithms. A survey showed that students were not satisfied with the interface. The latter was too dour and even though students were able to avoid some grammatical mistakes, the computer program did not prevent them all and did not teach students why they were about to make mistakes. The new version of AlgoPath now implements those features. With AlgoPath, students can no longer:

- Add a statement if it is not an output statement before a variable is declared;
- Add a variable in a R-value of an assignment if it was neither declared nor assigned a value;
- Assign a value to an aggregate or composite data type variable;
- Try to assign a variable if it was not declared;
- Forget or add the calling parameters of a module: the number of calling parameters are always equal to the parameters of the module;
- Associate a parameter of a prototype with a calling parameter of a different type;
- Use a composite data type if it is not declared;
- Forget what the basic data types are;
- Declare a variable if AlgoPath does not know the type;
- Forget what statements they can use;
- Assign a value that does not match the type of the variable.

Moreover, when students are about to make a mistake and AlgoPath is able to notice it, AlgoPath explains why it is a mistake and corrects it. In the future, we will focus on adding Object-oriented programming concepts such as objects, classes, data abstraction, encapsulation, polymorphism, and inheritance. We will also study the possibilities to add distributed algorithms and programming concepts.

Acknowledgment

We thank Dimitry Zekrouf, Nicolas Fleurentin, and Sonny Jestin for their valuable help during the study and the implementation of this new version of AlgoPath.

References

[1] E. Perrin, S. Linck, and F. Danesi, “Algopath: A new way of learning algorithmic,” in *The Fifth International Conference on Advances in Computer-Human Interactions*, Valencia, Spain, 2012.

[2] W. Barendregt and T. M. Bekker, “The influence of the level of free-choice learning activities on the use of an educational computer game,” *Computers and Education*, vol. 56, no. 1, pp. 80–90, 2011.

[3] M. Virvou and G. Katsionis, “On the usability and likeability of virtual reality games for education: The case of vr-engage,” *Computers and Education*, vol. 50, no. 1, pp. 154–178, 2008.

[4] M. Papastergiou, “Digital game-based learning in high school computer science education: Impact on educational effectiveness and student motivation,” *Computers and Education*, vol. 52, no. 1, pp. 1–12, 2009.

[5] D. B. Clark, B. C. Nelson, H.-Y. Chang, M. Martinez-Garza, K. Slack, and C. M. D’Angelo, “Exploring newtonian mechanics in a conceptually-integrated digital game: Comparison of learning and affective outcomes for students in taiwan and the united states,” *Computers and Education*, vol. 57, no. 3, pp. 2178–2195, 2011.

[6] C.-C. Liu, Y.-B. Cheng, and C.-W. Huang, “The effect of simulation games on the learning of computational problem solving,” *Computers and Education*, vol. 57, no. 3, pp. 1907–1918, 2011.

[7] B. Coller and M. Scott, “Effectiveness of using a video game to teach a course in mechanical engineering,” *Computers and Education*, vol. 53, no. 3, pp. 900–912, 2009.

[8] H. Tuzun, M. Yilmaz-Soylu, T. Karakus, Y. Inal, and G. Kizilkaya, “The effects of computer games on primary school students’ achievement and motivation in geography learning,” *Computers and Education*, vol. 52, no. 1, pp. 68–77, 2009.

[9] W. Yan, C. Culp, and R. Graf, “Integrating bim and gaming for real-time interactive architectural visualization,” *Automation in Construction*, vol. 20, no. 4, pp. 446–458, 2011.

[10] R. Hamalainen, “Designing and evaluating collaboration in a virtual game environment for vocational learning,” *Computers and Education*, vol. 50, no. 1, pp. 98–109, 2008.

[11] D. Charsky and W. Ressler, “”games are made for fun”: Lessons on the effects of concept maps in the classroom use of computer games,” *Computers and Education*, vol. 56, no. 3, pp. 604–615, 2011.

[12] Y.-T. C. Yang, “Building virtual cities, inspiring intelligent citizens: Digital games for developing students’ problem solving and learning motivation,” *Computers and Education*, vol. 59, no. 2, pp. 365–377, 2012.

[13] M. Ebner and A. Holzinger, “Successful implementation of user-centered game based learning in higher education: An example from civil engineering,” *Computers and Education*, vol. 49, no. 3, pp. 873–890, 2007.

[14] L. A. Annetta, J. Minogue, S. Y. Holmes, and M.-T. Cheng, “Investigating the impact of video games on high school students’ engagement and learning about genetics,” *Computers and Education*, vol. 53, no. 1, pp. 74–85, 2009.

[15] F.-L. Fu, R.-C. Su, and S.-C. Yu, “Egameflow: A scale to measure learners’ enjoyment of e-learning games,” *Computers and Education*, vol. 52, no. 1, pp. 101–112, 2009.

[16] D. J. Ketelhut and C. C. Schifter, “Teachers and game-based learning: Improving understanding of how to increase efficacy of adoption,” *Computers and Education*, vol. 56, no. 2, pp. 539–546, 2011.

Copyright (c) IARIA, 2013. ISBN: 978-1-61208-250-9

View publication stats