Microworlds for Learning Object-Oriented Programming: Considerations from Research to Practice

Fahima Djelil, Adélaïde Albouy-Kissi, Benjamin Albouy-Kissi, Eric Sanchez, Jean-Marc Lavest

Microworlds for Learning Object-Oriented Programming: Considerations from Research to Practice

Fahima Djelil, Adélaïde Albouy-Kissi, Benjamin Albouy-Kissi, Eric Sanchez and Jean-Marc Lavest

Author Note

Fahima Djelil, Adélaïde Albouy-Kissi, Benjamin Albouy-Kissi and Jean-Marc Lavest

Institut Pascal CNRS-UMR 6602, CNRS/UBP/SIGMA, Blaise Pascal University, Clermont-Ferrand, France

Eric Sanchez, French Institute of Education (IFE), Lyon, France. University of Fribourg, Switzerland.

Jean-Marc Lavest, French University in Armenia, Erevan, Armenia.

This research was supported by the French ministry of National Education, Higher Education and Research within Tactileo project

Correspondence concerning this article should be addressed to Fahima Djelil.

Institut Pascal, 4 Avenue Blaise Pascal, 63 178 Aubière, France.

E-mail: fahima.djelil@udamail.fr

Abstract

Object-Oriented paradigm is a common paradigm for introductory programming courses. However, many teachers find that transitioning to teaching this paradigm is a difficult task. To overcome this complexity, many experienced teachers use microworlds to give beginner students an intuitive and rapid understanding of fundamental abstract concepts of Object-Oriented Programming, leading to a more effective learning. Microworlds have thus, become known to be engaging and facilitating complex concepts understanding, however, we know little on the process by which microworlds can be effective for learning, and how they motivate beginners. In this article, we attempt to identify the design principles of microworlds that offer support structures for providing engaging and meaningful learning activities that facilitate programming learning. We will explore individually each design principle, while focusing on its relation to learning and its relevance for learning Object-Oriented Programming. We finally, present PrOgO, a novel programming microworld, we designed on the basis of existing microworlds.

Keywords: microworlds, object-oriented programming, game based learning, PrOgO

Introduction

Teaching Object-Oriented Programming (OOP) to beginners is widely known to be quite problematic (Börstler, Nordström, Westin, Moström, & Eliasson, 2008). Abstract concepts are the main difficulty students face in learning OOP. The concepts of object and class are the basic concepts that govern the Object-Oriented (OO) paradigm. As a type sets the properties of a variable, a class is often defined as a data structure representing a mold or a template for the object, while the object is an embodiment, a reproducible copy of its class (Bersini, 2007). The basics of OO paradigm comprise also the concept of hierarchical relationships among classes,

such as inheritance, which is a mechanism that allows factoring what is common to several classes into one class, in order to reduce the number of treatments.

Students struggle to understand more complex concepts, if they cannot grasp the basic class and object thinking way. Teaching introduction to programming is not about abstract OO concepts and practical programming skills alone, but about the interactions of both. Therefore, a teaching approach without balancing the two sides may not succeed in practice (Yan, 2009). According to Kölling & Rosenberg (2002), this difficulty of learning is not due to any complexity inner to OOP, but rather is in relation with the teaching methods and materials. They argued that many books authors often do not have teaching experience in the OOP paradigm, and that many development environments are not suited to this paradigm and fail to capture its full potential.

In order to understand how beginner students learn OOP, some computer science education researchers focus their attention on computer science teaching and particularly on teaching introductory programming. There exist many teaching introductory programming approaches, and since the mid-1990s, there has been considerable attention given to teaching OOP early (Bennedsen & Schulte, 2007).

The recent literature on introductory programming teaching is directed towards the use of microworlds (e.g., Becker, 2001; Cooper, Dann, & Pausch, 2003a; Cooper, Dann, & Pausch, 2003b; Henriksen & Kölling, 2004; Xinogalos, Satratzemi, & Dagdilelis, 2006). There are several reasons underlying interest of training professionals and educators in the use of microworlds. First, there has been a major shift in the field of learning, from a traditional model of instruction to a learner-centered model, emphasizing a more active learner role (Garris, Ahlers, & Driskell, 2002). A second reason is that some empirical evidence exists that

microworlds can be effective tools for enhancing learning and understanding complex programming concepts (e.g., Pears, et al., 2007; Bennedsen & Schulte, 2007). Third, training professionals are also interested in the involvement and engagement that microworlds can invoke (Pears, et al., 2007).

In brief, the potential of microworlds as tools for training is appealing. However, we know little on the process by which microworlds engage learners, and the types of learning outcomes that can be achieved through microworlds, in fact, few studies exist on the impact of microworlds on learning and education (e.g., Hogle, 1995; Rieber, 1996; Mavrikis, Noss, Hoyles, & Geraniou, 2013; Plass & Schwartz, 2014). Moreover, there may exists a risk of designing microworlds that neither instruct nor engage the learner. The purpose of this article is to explore the ways by witch microworlds can be relevant in introductory OOP courses. Our goal is to examine the aspects of microworlds that can enhance introductory programming learning. We focus on the conceptual foundations of microworlds that are of interest from an instructional perspective, and how they affect programming learning.

After introducing microworlds, we will examine each of their design principles. We will provide working definitions for each design principle, with a focus on its relation with learning in general and learning programming in particular. We will also present a novel programming microworld called PrOgO, we designed to help beginner students to learn the basics of OOP in the C++ programming language.

Defining Microworlds

Prior to dealing with these issues, it is useful to provide some definitions of what the term microworld means. According to Bruillard (1997), the term microworld had its genesis in the Winograd's SHRDLU program (Winograd, 1972), which operates within a small area (the

SHRDLU's world) to move various basic objects around in it. In the context of computer supported education, this term has been introduced for the first time by Minsky and Papert (1972) in an internal MIT research report. It has been later popularized by Papert with the LOGO programming language (Papert, 1980), which consists of a set of commands used to control a virtual or a robotic turtle within a small world.

According to Lawler and Lawler (1987), important functions of a microworld are the creation and manipulation of objects. Papert (1987) has described these objects as transitional between the objects that we can manipulate in the real world, and the objects that we know in science. They are a kind of objects that help to manipulate other abstract objects and serve as a link between intuitive learning and formal learning (Bruillard, 1997). Bruillard (1997) has provided perhaps the most comprehensive analysis of microworlds, describing a microworld as an artificial world in which the behavior of objects we manipulate adheres to some constraints of fidelity and consistency. The essential idea in this statement is the fidelity and the consistency of a certain model embodied in the microworld. In fact, Bruillard (1997) noted that microworlds establish a strong semantic link between something formal or abstract and something real, simulated or of reference, while maintaining the precise meaning. He proposed to schematize the various constituent parts of a microworld by distinguishing between a real world (a reference world) containing objects that are familiar to the learner, and a formal world (abstract world) containing science objects (see Figure 1). A modeling process or an exemplifying process relates these two worlds. The formal world is considered as a model of the real world, while the real world is an instance of the formal world. The learner can manipulate objects through an interface according to the underlying formal world, and their external aspects visualized through this interface recall the real world. Moreover, the Bruillard's general schema emphasizes the

existence of mental models on which microworlds are funded on. Bruillard (1997) describes a mental model as a representation in which symbolic objects behave similarly to the ones contained in the situations where they are represented. These mental models may change through the learner's experience within the proposed environments (Hogle, 1995; Rieber, 1996; Bruillard, 1997). Since microworlds embody a concept, the learner's role is to mentally represent this concept in order to grasp it (Bruillard, 1997).

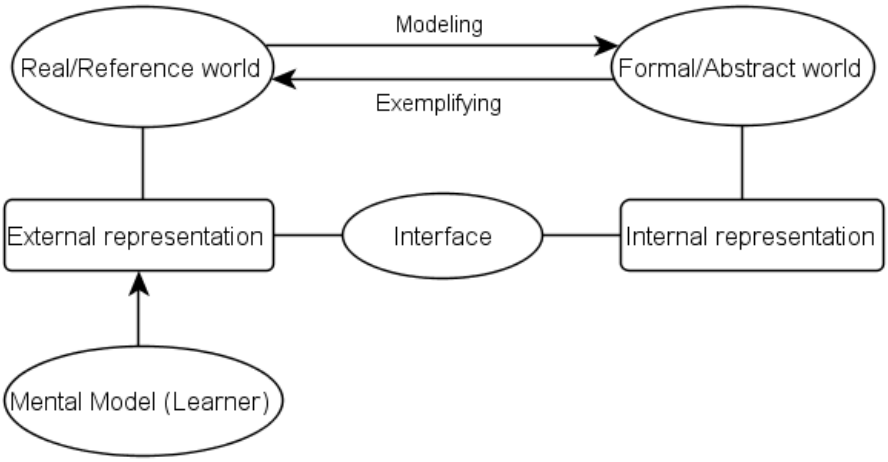

Figure 1. The Bruillard's general schema of microworlds (Bruillard, 1997).

Research and theory of mental models suggest that people form mental models of the physical world in an attempt to successfully understand and interact with the world (Rieber, 1996). Rieber (1996) reported three attributes of mental models relevant to microworlds design: a target system, which is the system of interest that comprises the science objects; the user's current mental model of the target system, which describes the person's current understanding of the target system; and a conceptual model of the target system, which is an artificial artifact. This confirms the Bruillard's definition of the microworlds' constituents (Bruillard, 1997). By referring to the Bruillard's general schema, the target system would correspond to the formal world, while the conceptual model would correspond to the reference world.

Rieber (2005) has later defined a microworld as an interactive digital environment that enables users to interact with models of situations and phenomena. This environment generates dynamic feedback based on a set of underlying rules, models, or computations as responses to the user's manipulations of objects and parameters (Rieber, 2005). This links up with the definition made by Hogle (1995), who considers a microworld as an interactive learning environment, which is a conceptual model of some aspects of the real world. An environment which is usually idealized and simplified, in which learners explore or manipulate the logic, rules, or relationships of the modeled concept, as determined by the designer (Hogle, 1995). It is therefore, defined as a cognitive tool, which is used as a means to simplify and design models of the real world, allowing students to manipulate and observe constraints and variables individually (Hogle, 1995).

Research into microworlds involves computer-based simulations and games (Hogle, 1995). However, microworlds, simulations and games are not synonymous, though in some cases they can considerably overlap (Hogle, 1995; Rieber, 1994). Simulations that are not different from real life experience, such as flight training simulators, are considered entirely distinct from microworlds (Hogle, 1995; Rieber, 1994). According to Rieber (1994), simulations start to become microworlds, when they are designed to let a novice begin to understand the underlying model. In order to take advantage from games, microworlds incorporate usually gaming techniques (Hogle, 1995; Rieber, 1996). Microworlds generally provide learners with game design as a way to learn about abstract subjects (Rieber, 1996). Learners are equipped to invent highly imaginative games to understand the world (Rieber, 1996). We would propose that microworlds contain game and simulation features, but they are neither games, nor simulations.

We argue in the following sections that there are four key dimensions that characterize microworlds: Metaphors, visualizations, constructivism and gaming. We later present PrOgO which we refer to as a microworld designed to help begginers to learn OOP in C++ programming language.

Programming Microworlds

The earliest and the most well known microworld that was designed to learn programming is LOGO (Papert, 1980). The LOGO microworld consists of a turtle geometry that allows learners access to principals of programming through interactive graphics. The LOGO statements manipulate a graphical turtle while a visual feedback includes geometric drawings such as lines, circles and squares (see Figure 2).

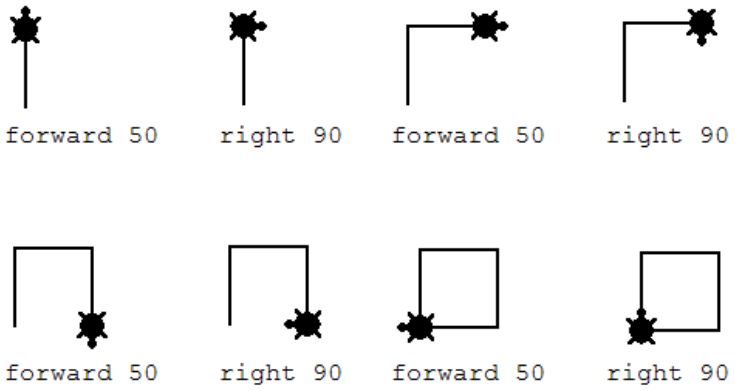

Figure 2. Examples of LOGO statements.

The majority of the programming microworlds that have been developed later are based on the same principle than Logo's system. Their main objective is to reduce the complexity of programming learning for novices. In the context of OOP, computing objects are given some of the properties of physical objects, such as appearance, behavior, and responsiveness to the user's actions, in order to be more easily graspable by novices. Abstract concepts are brought out to the user by means of a direct manipulation interface. Programming microworlds focus on concepts

rather than language syntax, and usually use subset of conventional programming languages (Cooper, Dann, & Pausch, 2000; Buck & Stucki, 2000). Moreover, they tend to engage students in learning programming by allowing them to write programs about games, stories and simulations (Cooper, Dann, & Pausch, 2000; Henriksen & Kölling, 2004). Representative OOP microworlds are Karel++ and its descendants, Alice and Greenfoot. Karel++ (Bergin, Stehlik, Roberts, & Pattis, 1997), Karel J Robot (Buck & Stucki, 2001; Bergin, Stehlik, Roberts, & Pattis, 2005), and ObjectKarel (Xinogalos, Satratzemi, & Dagdilelis, 2006) are programming learning environments that provide a display with a discrete 2D world of a robot called Karel, to which students can assign various programmable tasks (see Figure 3). Buck & Stucki (2001) noted that the power of Karel the robot is its ability to strip away details that are not important to the learned concepts. It rather allows students to build a more intuitive conceptual model (Buck & Stucki, 2001).

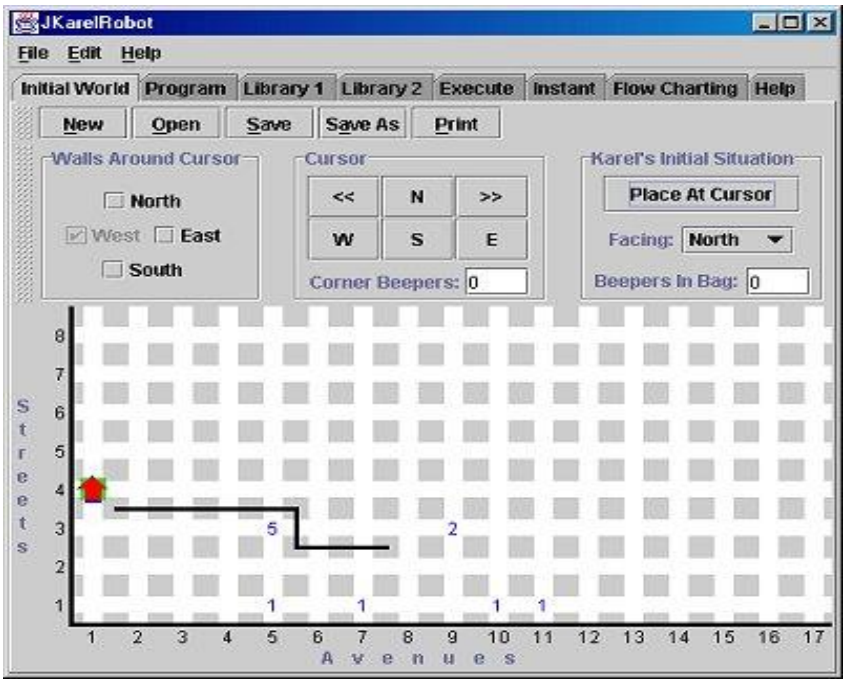

Figure 3. The Karel J Robot microworld.

Alice is a 3D interactive programming environment for building virtual worlds, animations, narrations and games. Alice allows students to populate a 3D virtual world with 3D models of objects, and control their appearance and behavior by writing simple scripts in a drag and drop editor, using drag and drop instructions (Cooper, Dann, & Pausch, 2000; Conway, Audia, Burnette, Cosgrove, & Christiansen, 2000) (see Figure 4).

Cooper, Dann, & Pausch (2003b) and Moskal, Lurie, & Cooper (2004) reported the use of Alice to teach programming for novices. They emphasized a design and a low number of details the students must master. They argued that working with an easy-to-use 3D graphics environment is attractive and highly motivating to today's generation of media conscious students. They observed that the visual nature and immediate feedback of program visualizations make it easy for students to see the impact of a statement or group of statements. Moreover, the 3D modeled classes and instantiated objects provide a very concrete notion of the concept of an object.

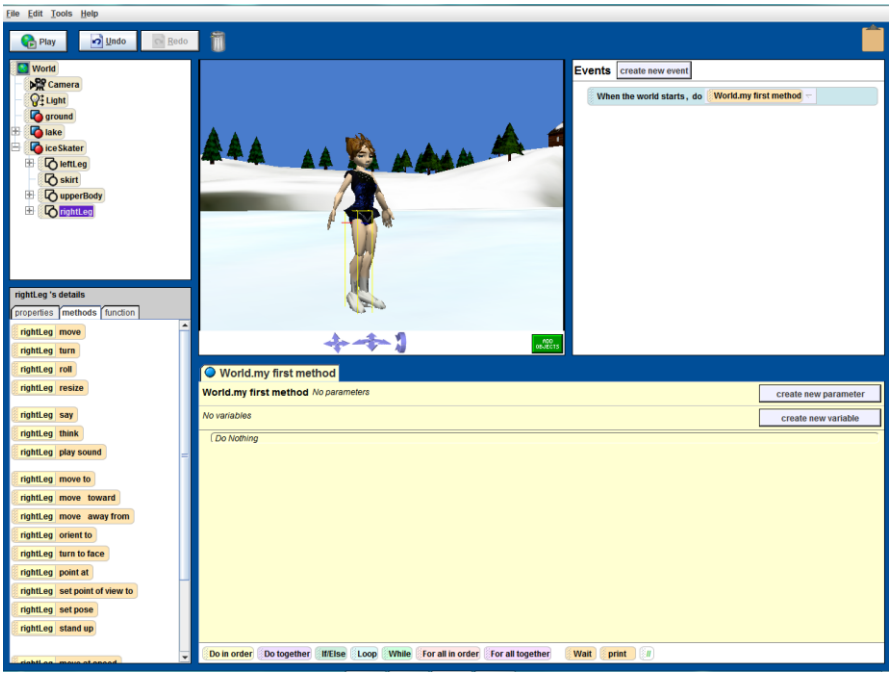

Figure 4. The Alice 2 microworld.

Greenfoot is an integrated development environment comprising a 2D world, in which students can place objects and program their behavior to create game scenarios and animations (Kölling, 2010) (see Figure 5). Greenfoot provides an API (Application Programming Interface) that offers a set of functionalities the student can use to create programs to control the animations inside the 2D scene. Greenfoot offers a display that includes a class diagram corresponding to the classes of the graphical objects represented in the 2D scene. A separate editor is also available allowing students to access to an existing code related to the available classes. This enables students to read a well-written code before creating code themselves.

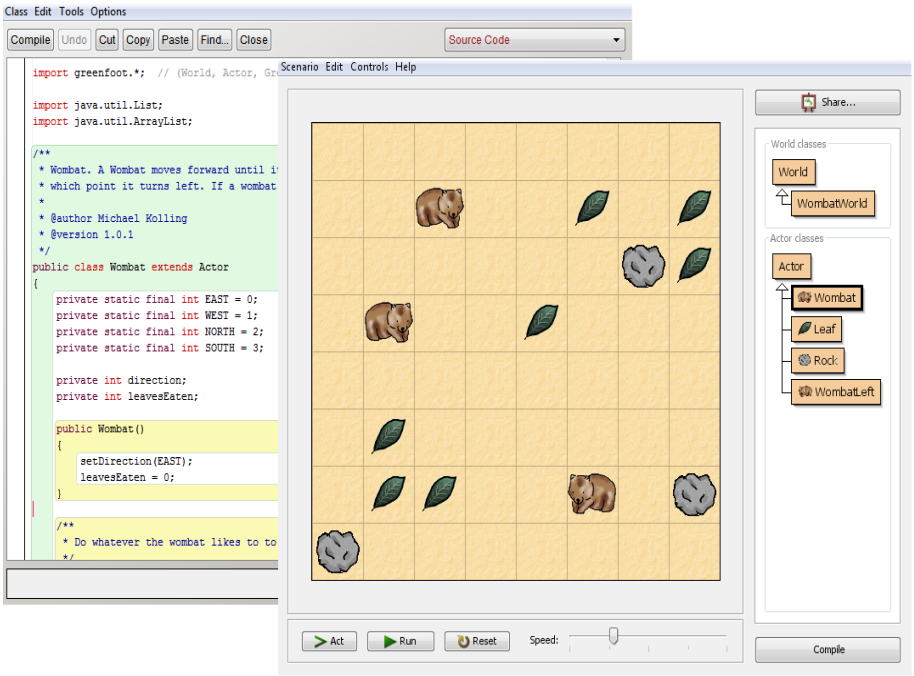

Figure 5. The Greenfoot microworld.

Characterizing Microworlds

Metaphors

Programming microworlds are built on metaphors. By the use of metaphors, programming microworlds intend to bring the abstract concepts of programming closer to the

real world, to be more easily graspable by beginners. This leads to decrease the distance between students' mental models and the programming language (Xinogalos, Satratzemi, & Dagdilelis, 2006). In such environments, a program is experienced as a model of some real world phenomenon, in which the concept of object is experienced as something that is active in a small world (a graphical scene), whereas a class is a description of properties and behavior of the object (see Cooper, Dann, & Pausch, 2000; Henriksen & Kölling, 2004; Bergin, Stehlik, Roberts, & Pattis, 2005).

According to Travers (1996), computation itself is a structuring metaphor and programming models are built on metaphors. This includes the OOP paradigm, in which computational objects are depicted metaphorically in terms of physical and social objects. Like physical objects, they can have properties and state, and like social objects, they can communicate and respond to communications (Travers, 1996).

Historically, OOP arose out of languages designed for simulation, particularly Simula (Kirkerud, 1989) and for novice programming in graphic environments such as Smaltalk (Sharp, 1996), in which the computational objects represent real-world objects. A standard example is a spaceship, which is modeled by a computational object that has properties like position, orientation and mass, and can perform actions like rotate and accelerate. The OO metaphor explicitly represents relationship between computational structures and real world objects. This has been reflected in the actual OOP microworlds, which make computational objects real for the user, making them appear to have a solid existence, having the properties of physical and tangible objects and being accessible to the actions of the user (Conway, Audia, Burnette, Cosgrove, & Christiansen, 2000; Bergin, Stehlik, Roberts, & Pattis, 2005; Kölling, 2010).

Several studies have been conducted in order to determine the impact of metaphors on learning. Carroll & Mack (1999) reported that most educators have often observed that providing students with comparisons can help them learn, and that metaphors serve to codify and communicate new knowledge in a comprehensible way for new learners. In computer science teaching, most important computing developments are funded on metaphors, therefore, explicit metaphors are often used to teach beginner students how to program (McConnell, 2004). In this field, the term metaphor is used not in a narrow linguistic sense, but in the sense of rich and complex metaphoric models (Travers, 1996). A metaphoric model is defined as a way to structure the knowledge of one domain by mapping onto it concepts and relations from an existing domain that is already familiar (Lakoff & Johnson, 2008). Metaphor in this sense, consists of a fundamental way of learning and a structuring conceptual system (Lakoff & Johnson, 2008).

Since the purpose of metaphoric models is to describe something that is hidden in terms of something visible, metaphors are often taken from the concrete physical world. Metaphors make programming tangible, since they provide students means to understand the abstruse operations of the computer in terms borrowed from more familiar domains (Travers, 1996). This is almost very useful for learning about concepts which can not be directly perceived. Mayer (1975) has experimentally studied the effectiveness of metaphors in teaching programming concepts. He noted that many programming constructs could be learned more easily when they are presented in the context of a concrete metaphor (Mayer, 1975). He later observed that giving beginners metaphorical models of computer language interpreters, resulted in improved learning compared to a more literal technical presentation (Mayer, 1981). He concluded that familiarity with the model aids in the assimilation of technical content by giving it a meaningful context.

Another important aspect of programming microworld metaphors is the use of animations, which have proven to be a powerful conceptual tool for analyzing programming paradigms (Travers, 1996). Animate metaphors are known to be powerful because they are capable of exploiting our knowledge about humans and human actions (Travers, 1996).This allows to think in terms of purpose and functions, and thus to understand a system, without having to know its implementation details.

Visualizations

Programming microworlds are built within a design that explicitly visualizes fundamental concepts of programming in a realistic and a meaningful context. Direct manipulation of graphics representing OO concepts can help students, as in real life, to think in terms of objects. Students do not typically start by manipulating source code, they rather start by creating objects by selecting graphics (see Cooper, Dann, & Pausch, 2000; Kölling, 2010).

Gilbert (2010) defined a visualization as a visual representation which can be internal to an individual (known as a mental image or a personal mentally construction), or external (open to inspection by others). He claimed that the existence of all visualizations depends on the operation of metaphor and analogy, since we initially tend to think of something new in terms of something with which we are more familiar. Metaphors and analogies are very used to explain complex concepts or to illustrate processes, since visual metaphors are arguably much more powerful than verbal ones (Baldwin & Kuljis, 2001). According to Gilbert (2010), representations are the entities with which all thinking is considered to take place. They serve as means to depict the models that we have created so that the individual concerned can perceive what has been done and can share that with others. They are therefore important in the conduct and learning of science (Gilbert, 2010).

Visualization is largely used to graphically illustrate various concepts in computer science (e.g., Esteves & Mendes, 2003; Milne & Rowe, 2004; Moreno, Myller, Sutinen, & Ben-Ari, 2004; Sajaniemi, Byckling, & Gerdt, 2007; Osman & Elmusharaf, 2014). The impetus for visualization in computing comes from the inherent abstractness of the basic concepts. OOP paradigm is itself an especially natural foundation for visualization (Jerding & Stasko, 1994), since it fundamentally involves the manipulation of concrete things: instances, messages, methods, and so on. Thus, building visualization for OO concepts follows naturally from their correspondence to a visual representation. By making these concepts more concrete, graphical representations help to better understand how they work. Naps, et al., (2002) claimed that using visualizations can positively impacts learning and thus help students to learn computing concepts. Henriksen & Kölling (2004) have moreover observed that visualization and interactivity in a learning environment allow experimentation, generate curiosity, and thus create student engagement.

Programming microworlds are developed with a view to make programming more accessible to novices, enabling learners to program by building and manipulating animated visual representations of programming concepts. To make computational objects real for novices, who have not yet developed the skills to mentally visualize them. Objects are reified so that the result of command execution is visible as the position, size, rotation, and other visible state and behavior of the object changes (see Cooper, Dann, & Pausch, 2000; Xinogalos, Satratzemi, & Dagdilelis, 2006 ; Kölling, 2010). The purpose of such design is to allow the user to think of computational objects as identical with their graphic representations, and thus to structure a correct understanding of what programming concepts consist of.

Constructivism

Microworlds are based on the constructivist learning theory, as indicated by several authors : Microworlds represent an immediate application for the infusion of constructivism into instructional design (Rieber, 1992); microworlds are interactive environments in which we can learn about a specific domain through personal discovery and exploration (Papert, 1980); the power of microworlds as learning tools is based in the philosophy of constructivism (Bliss & Ogborn, 1989).

Constructivism assumes that we learn best when we actively construct ideas and relationships in our minds based on experiments we do, rather than being told (Nix & Spiro, 1990), and that we learn with particular effectiveness when we are engaged in constructing personally meaningful physical artifacts (Prensky, 2005). Constructivism essentially consists of the idea that learning involves individual constructions of knowledge. Microworlds push this responsibility further by helping learners to determine the correctness of their own solutions, rather than reserving this responsibility for a teacher (Hogle, 1995). Microworlds succeed in that, since they offer a graphic and quick feedback leading to self-correction.

Microworlds rely on constructivism, since they are based on visualizations applying metaphors that help novices to construct knowledge and skills by themselves. As it was claimed by Carroll & Mack (1999), metaphors' primary function is to stimulate active learning. The use of metaphors facilitate active learning by providing clues for logical inferences through which learners construct their knowledge. Microworlds are therefore, intrinsically based on constructivism.

All this is in concordance with the proposition made by Hadjerrouit (2005), who has defined a pedagogical framework rooted in a constructivist epistemology for teaching OO design

and programming. He suggested that OO knowledge must be actively constructed by learners, rather than being passively transmitted by teachers. Program development must be guided by OO concepts, rather than language technicalities, and traditional modes of teaching such as lectures, must be replaced by a set of activities where the students are actively engaged in the knowledge being constructed. The activities must focus around a set of realistic and intrinsically motivating problems. These principles fit into the goals of microworlds, constructivism is therefore one of their instructional pillars.

Gaming

Another feature of microworlds is their incorporation of gamelike elements. Nowadays, games become popular implementations of microworlds (Cooper, Dann, & Pausch, 2003a; Rieber, 2005; Kölling, 2010; Plass & Schwartz, 2014). The belief is that students' familiarity with games can be used to motivate computer science learning and attract future generations of computer scientists (Tapia, El-Nasr, Yucel, Zupko, & Maldonado, 2007). Usually, in the context of OOP microworlds, learners are provided with game design as a way to learn about abstract programming concepts. Examples of OOP microworlds that use game design and building as a learning activity are typically Alice (Conway, Audia, Burnette, Cosgrove, & Christiansen, 2000) and Greenfoot (Kölling, 2010), which allow students to create game scenarios by making programs.

Characteristics of games may provide some practical means of meeting the assumptions of a successful microworld. One important characteristic of microworlds is a design principle that provides learning activities that are intrinsically motivating (Papert, 1980). Games represent the instructional artifacts most matching this characteristic (Garris, Ahlers, & Driskell, 2002). Motivational researchers have linked intrinsic motivation to a number of other characteristics.

Malone (1982) proposed that the primary factors that make an activity intrinsically motivating are challenge, curiosity, and fantasy. Such characteristics are widely applied in the design of instructional games (Garris, Ahlers, & Driskell, 2002; Dondlinger, 2007). Fantasy is used to encourage learners to imagine that they are completing activities inside an imaginary world having no impact on the real world (Garris, Ahlers, & Driskell, 2002). Challenge and curiosity closely conform to the process of equilibrium (Rieber, 1996). When confronted with a problem without an immediate solution, a learner will seek resolution if a solution seems possible. Learners are challenged by activities that are neither too easy nor too difficult. Feedback can also easily be provided in order for learners to quickly evaluate their progress against the established game goal (Garris, Ahlers, & Driskell, 2002; Rieber, 1996).

Games have also the potential to meet the microworld's goal of constructivism learning approach. As stated by several authors: Constructivist learning theory can be guided using games (Papert, 1980); by building games, students learn programming within a constructivist approach (Kafai, 2006). Students are actively engaged in discovering and building their own understanding of new concepts and skills, when they design their own programs to create games. Therefore, games provide a good environment for promoting different aspects of microworlds including active learning, self-regulation and intrinsic motivation.

PrOgO: a novel microworld for learning Object-Oriented Programming

PrOgO is a programming microworld, which is based on a constructive game metaphor for learning the basics of OOP in C++ programming language for beginners. We initially referred to the experimental prototype of PrOgO as a 3D virtual game, since it is based on a constructive game metaphor (see Djelil, Albouy-Kissi, Albouy-Kissi, Sanchez, & Lavest, 2015). However, PrOgO does not sufficiently incorporate game elements such as high levels of fantasy,

mystery and challenges. It is rather designed as a programming microworld, which aims to help students to access easily to the conceptual model triggering the OOP paradigm.

The PrOgO gaming objective is to build and animate 3D graphical robots or mechanical structures, whereas each 3D elementary graphic corresponds to a computing object. Based on the principle that a computing object is an active entity within a program, including a set of attributes (knowledge) and methods (skills) that act to achieve some functionalities that constitute the program's objectives, we assume that a constructive game metaphor would be relevant to introduce OOP concepts to beginners. Like a physical block, a computing object has attributes that define its appearance (a position specified in x, y, z coordinates, a rotation, a translation and a color). Like a constructive game piece, a computing object can be connected to another one, to constitute a more complex structure that has enough skills and knowledge to be able to perform actions in a giving environment.

PrOgO provides students with a direct manipulation interface comprising realistic graphics representing OOP concepts, to help them to think about objects as in real life. Students can create objects, visualize their state and behavior, interact with them by changing the values of their attributes, executing their methods and make them communicate with one another. In order to help beginners understanding more easily the implementation of abstract concepts in a specific syntax such as C++, PrOgO interpretes into source code, all the manipulations executed directly on the 3D graphics. Since the visual representations are very close to the abstract concepts, each student's action is automatically interpreted into a C++ statement. This will constitute a complete program that the student can visualize and modify in order to practice programing.

The PrOgO interface (see Figure 6) comprises, in its center, a 3D scene in which there is permanently a 3D block called base, providing a construction base to witch objects that are being created will be linked. On the lower left corner of the interface, there is the window Classes, which comprises all the 3D elementary blocks a student can instantiate to obtain objects. The top left of the interface displays the tree structure of the objects that have been created and assembled to each other. A code completion editor is anchored on the right side of the 3D scene, to display the C++ code that is being generated during the direct manipulations of the graphics, and allow students to implement the addressed concepts, while avoiding syntax errors. Once an object is created and placed into the 3D scene, the student can view all its attributes and methods. Once an attribute value is changed, or a method is executed, its result on the state of the object is immediately visualized and the corresponding C++ code is automatically generated. Similarly, once a statement is completed in the code editor its result on the concerned object is immediately visualized in the 3D scene.

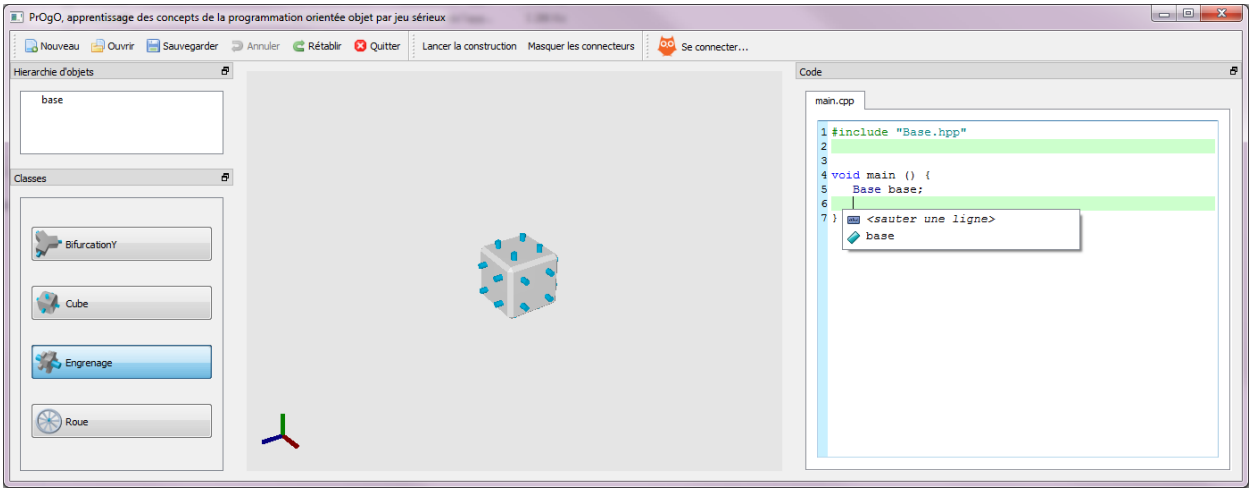

Figure 6. The PrOgO main window.

Conclusion

Microworlds are cognitive learning tools that are used to promote abstract concepts understanding. They are small, interactive, and contain programmable models of real world environments. They are engaging and appealing for students learning based on constructivist principles: Active participation, individual knowledge construction, discovery and meaningful learning contexts. Microworlds are based on knowledge visualizations that aim at augmenting communication via graphics means that mediate knowledge through metaphors. In the context of OOP, they expand beyond programming languages to allow learners to explore and build models through the direct manipulation of objects on screen. They include game design elements promoting intrinsic motivation, self-regulation and active learning.

Research suggests that strategies involving active learning by the use of metaphors, visualizations, and gaming are excellent ways for people to explore a domain in rich and meaningful ways (Rieber, 1996; Carroll & Mack 1999; Naps, et al., 2002). These attributes constitute the pillars that guide the design of microworlds in which they are carefully blended. Such design provides an effective learning strategy that allows students to master learning concepts, increase their motivation and optimize the ownership of knowledge.

This study allowed to throw light upon the potential of microworlds in introducing OOP concepts, and their learning effectiveness by exploring their design foundations. We concluded that the power of microworlds as learning environments, lies on their design principles that are very rich, leading to an effective learning.

Microworlds succeed as learning environments in introductory OOP courses, since they foster the individual construction of a correct mental model of what is OOP paradigm. This occurs, as microworlds use concrete metaphors that have the potential to help novices to perceive

very easily abstract and complex concepts. Using visualizations helps students to think of computational objects as identical with their graphic representations. This is mainly what allows to reify the metaphoric model witch dictates the OOP paradigm, that becomes more easy to be graspable by novices.

Moreover, microworlds succeed as engaging environments, as they incorporate gamelike elements. Games offer many advantages to microworlds design by having the potential to meet most of microworlds' objectives. Games provide characteristics of intrinsic motivation and foster the constructivist nature of microworlds.

Ultimately, this study allowed us to identify significant mediating variables in the design of a new microworld. It was therefore critically important, to choose the right metaphor that can allow beginner students to access easily the conceptual model of the OOP paradigm, and thus to be engaging and effective in introductory OOP courses.

Acknowledgement

This work has been financed by the French Ministry of National Education, Higher Education and Research within the Tactileo project. The authors would like to thank Delphine Huguel for her work in the development of PrOgO.

References

Baldwin, L. P., & Kuljis, J. (2001). Learning programming using program visualization techniques. *Proceedings of the 34th Annual Hawaii International Conference on System Sciences* (p. 8).

Becker, B. W. (2001). Teaching CS1 with Karel the robot in Java. *ACM SIGCSE Bulletin, 33*(1), 50-54.

Bennedsen, J., & Schulte, C. (2007). What does objects-first mean?: An international study of teachers' perceptions of objects-first. *Proceedings of the Seventh Baltic Sea Conference on Computing Education Research-Volume 88* (pp. 21-29). Australian Computer Society.

Bergin, J., Stehlik, M., Roberts, J., & Pattis, R. (1997). *Karel++: A gentle introduction to the art of Object-Oriented Programming.* Wiley.

Bergin, J., Stehlik, M., Roberts, J., & Pattis, R. (2005). *Karel J Robot: A gentle introduction to the art of Object-Oriented Programming in Java.* Dream Songs Press.

Bersini, H. (2007). *L'orienté objet: cours et exercices en UML 2 avec Java 5, C Sharp 2, C++, Phyton et PHP 5.* In Eyrolles (Ed.).

Bliss, J., & Ogborn, J. (1989). Tools for exploratory learning: A research program. In W. O. Library, (Ed.), *Journal of Computer Assisted Learning, 5*(1), 37-50.

Börstler, J., Nordström, M., Westin, L. K., Moström, J.-E., & Eliasson, J. (2008). Transitioning to OOP/Java — A never ending story. In J. Bennedsen, M. Casperson, & M. Kölling, *Reflections on the teaching of programming* (pp. 80-97). Springer.

Bruillard, E. (1997). *Les machines à enseigner.* In Hermès (Ed.).

Buck, D., & Stucki, D. J. (2000). Design early considered harmful: graduated exposure to complexity and structure based on levels of cognitive development. In ACM (Ed.), *ACM SIGCSE Bulletin, 32*(1), 75-79.

Buck, D., & Stucki, D. J. (2001). JKarelRobot: a case study in supporting levels of cognitive development in the computer science curriculum. In ACM (Ed.), *ACM SIGCSE Bulletin, 33*(1), 16-20.

Carroll, J. M., & Mack, R. L. (1999). Metaphor, computing systems, and active learning. In Elsevier (Ed.), *International Journal of Human-Computer Studies, 51*(2), 385-403.

Conway, M., Audia, S., Burnette, T., Cosgrove, D., & Christiansen, K. (2000). Alice: Lessons learned from building a 3D system for novices. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 486-493.

Cooper, S., Dann, W., & Pausch, R. (2000). Alice: a 3-D tool for introductory programming concepts. *Journal of Computing Sciences in Colleges, 15*(5), 107-116.

Cooper, S., Dann, W., & Pausch, R. (2003a). Teaching objects-first in introductory computer science. *ACM SIGCSE Bulletin, 35*(1), 191-195.

Cooper, S., Dann, W., & Pausch, R. (2003b). Using animated 3D graphics to prepare novices for CS1. *Computer Science Education, 13*(1), 3-30.

Djelil, F., Albouy-Kissi, B., Albouy-Kissi, A., Sanchez, E., & Lavest, J.-M. (2015). Towards a 3D virtual game for learning Object-Oriented Programming fundamentals and C++ launguage. Theoretical considerations and empirical results. *7the International Conference on Computer Supported Education (CSEDU)* .

Dondlinger, M. J. (2007). Educational video game design: A review of the literature. *Journal of applied educational technology, 4*(1), 21-31.

Esteves, M., & Mendes, A. (2003). OOP-Anim, a system to support learning of basic object oriented programming concepts. *Proceedings of CompSysTech'2003-International Conference on Computer Systems and Technologies.*

Garris, R., Ahlers, R., & Driskell, J. E. (2002). Games, motivation, and learning: A research and practice model. In S. Publications (Ed.), *Simulation & Gaming, 33*(4), 441-467.

Gilbert, J. K. (2010). The role of visual representations in the learning and teaching of science: An introduction. *Asia-Pacific Forum on Science Learning and Teaching, 11*(1), 1-20.

Hadjerrouit, S. (2005). Object-oriented software development education: a constructivist framework. *Informatics in Education-An International Journal, 4*(2), 167-192.

Henriksen, P., & Kölling, M. (2004). Greenfoot: combining object visualization with interaction. *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, 73-82.

Hogle, J. G. (1995). Computer microworlds in education: Catching up with Danny Dunn. *ERIC*.

Jerding, D. F., & Stasko, J. T. (1994). *Using visualization to foster object-oriented program understanding.* Graphics, Visualization & Usability Center, Georgia Institute of Technology.

Kafai, Y. B. (2006). Playing and making games for learning instructionist and constructionist perspectives for game studies. In S. Publications (Ed.), *Games and culture, 1*(1), 36-40.

Kirkerud, B. (1989). *Object-oriented programming with SIMULA.* Addison-Wesley Longman Publishing Co., Inc.

Kölling, M. (2010). The Greenfoot programming environment. In ACM (Ed.), *ACM Transactions on Computing Education (TOCE), 10*(4), 14.

Kölling, M., & Rosenberg, J. (2002). *BlueJ-The Hitch-Hikers guide to object orientation.* USD.

Lakoff, G., & Johnson, M. (2008). *Metaphors we live by.* University of Chicago press.

Lawler, R. W., & Lawler, G. P. (1987). Computer microworlds and reading: An analysis for their systematic application. *Artificial intelligence and education*, 95-115.

Malone, T. (1982). What makes computer games fun? In ACM (Ed.), *13*(2-3).

Mavrikis, M., Noss, R., Hoyles, C., & Geraniou, E. (2013). Sowing the seeds of algebraic generalization: Designing epistemic affordances for an intelligent microworld. In W. O. Library (Ed.), *Journal of Computer Assisted Learning, 29*(1), 68-84.

Mayer, R. E. (1975). Different problem-solving competencies established in learning computer programming with and without meaningful models. In A. P. Association (Ed.), *Journal of Educational Psychology, 67*(6), 725.

Mayer, R. E. (1981). The psychology of how novices learn computer programming. *ACM Computing Surveys (CSUR), 13*(1), 121-141.

McConnell, S. (2004). *Code Complete: A Practical handbook of software construction.* Microsoft Press.

Milne, I., & Rowe, G. (2004). Ogre: Three-dimensional program visualization for novice programmers. *Education and Information Technologies, 9*(3), 219-237.

Minsky, M., & Papert, S. (1972). *Artificial intelligence progress report.* Massachussetts Institute of Technology *(MIT).*

Moreno, A., Myller, N., Sutinen, E., & Ben-Ari, M. (2004). Visualizing programs with Jeliot 3. In *Proceedings of the working conference on Advanced visual interfaces*, 373-376.

Moskal, B., Lurie, D., & Cooper, S. (2004). Evaluating the effectiveness of a new instructional approach. *ACM SIGCSE Bulletin, 36*(1), 75-79.

Naps, T. L., Röβling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., . . . Rodger, S. (2002). Exploring the role of visualization and engagement in computer science education. *ACM SIGCSE Bulletin, 35*(2), 131-152.

Nix, D., & Spiro, R. J. (1990). *Cognition, education, and multimedia: Exploring ideas in high technology.* Routledge.

Osman, W. I., & Elmusharaf, M. M. (2014). Effectiveness of combining algorithm and program animation: A case study with data structure course. *Issues in Informing Science and Information Technology*.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas.* Basic Books, Inc.

Papert, S. (1987). Microworlds: Transforming education. *Artificial intelligence and education*, 79-94.

Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., . . . Paterson, J. (2007). A survey of literature on the teaching of introductory programming. *ACM SIGCSE Bulletin, 39*(4), 204-223.

Plass, J. L., & Schwartz, R. N. (2014). Multimedia learning with simulations and microworlds. *Cambridge handbook of multimedia learning*, 729-761.

Prensky, M. (2005). Computer games and learning: Digital game-based learning. In M. MIT Press Cambridge (Ed.), *Handbook of computer game studies, 18*, 97-122.

Rieber, L. P. (1992). Computer-based microworlds: A bridge between constructivism and direct instruction. *Educational technology research and development, 40*(1), 93-106.

Rieber, L. P. (1994). *Computers, graphics & learning.* Brown & Benchmark Madison, WI.

Rieber, L. P. (1996). Seriously considering play: Designing interactive learning environments based on the blending of microworlds, simulations, and games. *Educational technology research and development, 44*(2), 43-58.

Rieber, L. P. (2005). Multimedia learning in games, simulations, and microworlds. *The Cambridge handbook of multimedia learning*, 549-567.

Sharp, A. (1996). *Smalltalk by example: The developer's guide.* McGraw-Hill, Inc.

Sajaniemi, J., Byckling, P., & Gerdt, P. (2007). Animation metaphors for object-oriented concepts. *Electronic Notes in Theoretical Computer Science, 178*, 15-22.

Tapia, A. H., El-Nasr, M. S., Yucel, I., Zupko, J., & Maldonado, E. (2007). Building virtual spaces. *Virtuality and Virtualization*, 317-334.

Travers, M. D. (1996). *Programming with Agents: New metaphors for thinking about computation.* Citeseer.

Winograd, T. (1972). Understanding natural language. *Cognitive psychology*, *3*(1), 1-192.

Xinogalos, S., Satratzemi, M., & Dagdilelis, V. (2006). An introduction to object-oriented programming with a didactic microworld: objectKarel. *Computers & Education*, *47*(2), 148-171.

Yan, L. (2009). Teaching object-oriented programming with games. *Sixth International Conference on Information Technology: New Generations*, 969-974.