Rapid Digital Game Creation for Learning Object-Oriented Concepts

***Nikunj Dalal** (Management Science and Information Systems),*
***Subhash Kak** (Computer Science), and*
***Sohum Sohoni** (Computer Engineering)*
Oklahoma State University, Stillwater, OK, USA

nik@okstate.edu; subhash.kak@okstate.edu;
sohum.sohoni@okstate.edu

Abstract

A large body of research from multiple fields demonstrates the power of digital games in learning. This article is about the learning that occurs from making games, rather than from playing games. In this paper, we describe the use of Rapid Digital Game Creation (RDGC) for learning and teaching Object-Oriented (O-O) concepts. RDGC involves the rapid building of digital games with high-level software that requires little or no programming knowledge. We examine how RDGC supports the understanding of various O-O concepts. Using a theoretical framework of constructionism, we discuss pedagogical guidelines for RDGC-based learning. We suggest that RDGC is a useful pedagogic tool that complements formal programming languages and can help flatten the steep learning curve needed to learn O-O computer programming (or OOP).

Keywords: Rapid digital game creation, Learning, Object-oriented concepts, Pedagogy, Computing education

Introduction

In recent years, digital game based learning has received considerable attention from researchers as it has been found that playing videogames can enhance learning in both adults and children. A large body of research from multiple fields (Babcock & Marks 2010; Gee 2007; Lenhart, Kahne, Middaugh, Macgill, Evans, & Vitak, 2008) demonstrates the power of games, and a growing number of researchers are incorporating games into education. Digital games are linked to excitement, energy, motivation, imagination, learning, and flow. Play fosters learning, flexibility, and creativity (Silveira, Araújo, Veiga, Naito, & Comotti, 2011). Moreover, recent research (Lenhart et al., 2008) has shown that games cut across gender, ethnic, and socio-economic boundaries. This study showed how ubiquitous games are to the younger generation raised in the computer, gaming and Internet era, that is, the digital natives (Prensky, 2001) — 98% of teenage boys and 94% of girls play electronic games—and shattered stereotypes about games as being restricted to a "solitary, nerd" subculture. The motivational potential of digital games staggers the imagination. Games form a context for learning which can be understood by people from diverse cultures and backgrounds.

Considerable research in digital game based learning has been about the learning that occurs from *playing* games. But what is the learning that can occur from *building* rather than playing games? Curricula that have used game design in computing currcula have largely found high motivation, increased learning, and positive effects on students (e.g., Bayliss & Strout, 2006; Parberry, Kazemzadeh, & Roden, 2006). In recent years, the dramatic increase in rapid prototyping tools for building games and applications is leading to a new interest in innovative learning from rapidly creating games. Hence, in this paper, we specifically focus on *rapid* game creation. Rapid Digital Game Creation or RDGC is the process "used to build computer games quickly and easily using game creation software that requires little or no programming knowledge. Rapid game creation enables a creator to build a quick prototype game and to see the effects of changes almost immediately (Dalal, Dalal, Kak, Antonenko, & Stansberry, 2009, p. 125)." Why focus on learning from rapid game creation and not from design of digital games from scratch? Design of games from scratch is a very complex cognitive activity (Sweller, 1998) involving considerable expertise, high costs, and development time, and as such is not readily possible or available for wide communities of learners. In contrast, RDGC-based learning can be a fun process that is not highly difficult or expensive to implement and maintain.

While RDGC-based learning involves fun, motivation, art, music, and creativity, our focus in this paper is on the use of RDGC for learning and teaching of object-oriented concepts. All major academic recommendations for information systems and computing curricula (http://www.acm.org/education/curricula-recommendations) include the concepts of O-O programming, O-O analysis, O-O modeling, and O-O design. Concepts such as classes, objects, events, instances and their subconcepts are intrinsic to the understanding of the development of modern systems. Moreover, such concepts make up a deeper type of thinking skill important for students to learn: that we might call object-oriented thinking. As Wright (2007) asserts:

> "Object-oriented thinking has been around even before object-oriented programming. People do it without knowing it might be called object-oriented. It helps us conceptualize a system and better grasp it. It helps us wrap our mind around a system without blowing a fuse. It makes programming easier for us and easier for others coming to our code. Object-oriented programming was created to make it easy to transfer our object-oriented thinking into code, although we can still program procedurally our object-oriented design."

Object-oriented thinking, in turn, may be seen as an aspect of computational thinking, which "represents a universally applicable attitude and skill set that everyone, not just computer scientists, would be eager to learn and use…It is concerned with conceptualizing, problem-solving and designing systems drawing upon mathematical and engineering thinking using concepts fundamental to computing" (Wing, 2006).

In this paper, we propose the use of RDGC for the learning and innovative teaching of basic and advanced O-O concepts. This paper is organized as follows. In the next section, we describe the RDGC process and tools and demonstrate the creation of a Pong game using a tool called Game Maker. Next, we discuss the theoretical basis of constructionism. Then we discuss pedagogy for O-O learning and teaching with RDGC. Finally, we conclude with limitations and implications for future teaching and research.

Rapid Digital Game Creation

As described earlier, RDGC refers to the process of building computer games quickly and easily, using game creation software that requires little or no programming knowledge. RDGC offers an easy and enjoyable way of achieving this task of building computer games. It does not require the user to have prior knowledge of programming. There are various RDGC tools available such

as Game Maker (http://www.yoyogames.com), Multimedia Fusion (http://www.clickteam.com/website/usa), Alice (http://www.alice.org), and Scratch (http://scratch.mit.edu/) among others. Tools such as App Inventor (http://www.appinventorbeta.com/) are also available for building mobile games and apps. The tools vary in several aspects such as ease of use, ease of learning, type of deployment platform, and in the availability of different complex options for building a game. But they all offer a visual object oriented platform with a variety of options to create and specify objects, events and methods. Some tools are specialized for building games whereas others can be used for building more general applications or software. However, note that the approach described in this paper is platform independent as any of the tools can be used as a means to learn and teach O-O skills.

As an example, see Figure 1, which shows the user-interface of Multimedia Fusion 2. Area 1 is the workspace that shows the levels of the game. Area 2 shows the properties of a game or any of its objects. The objects used by the game are themselves displayed in Area 3. These objects can be laid out in a frame as shown in Area 4. Other rapid game-making programs have interfaces with similar functionality.

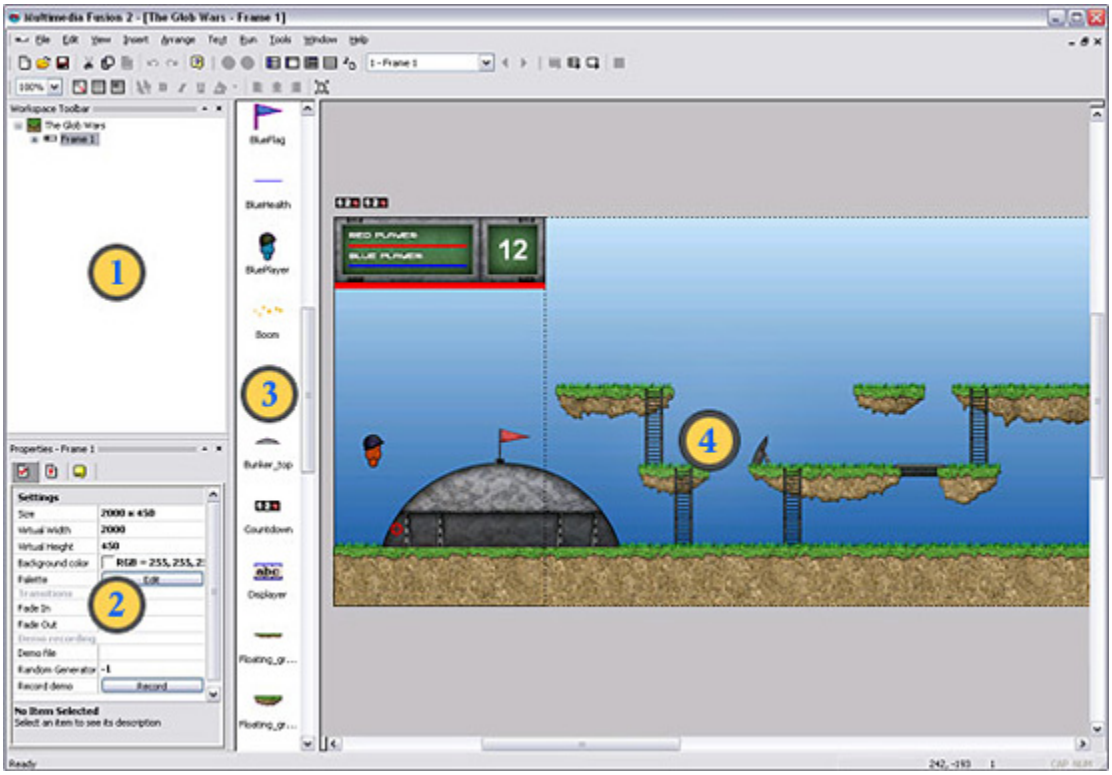

Figure 1: Screenshot of the user interface of Multimedia Fusion 2

Creating a Game in Game Maker

In this paper, we illustrate the RDGC process in relation to O-O concepts using Game Maker as the platform because it is popularly available in the public domain and because of the relatively short learning curve it requires (Habgood & Overmars, 2006). Game Maker enables the game creator to create sprites (the graphic images for the characters required for the game), objects (where the characters created using the sprites are assigned properties), events (e.g., collision be-

tween objects), actions in response to the events (e.g., ball goes up upon collision with the wall), rooms (used to implement different levels of the game), sounds, backgrounds, and others.

Figure 2 shows a screen shot from a prototype Pong game created using Game Maker. The Pong game was designed as an aid to illustrate O-O concepts. The time it takes to create the initial game is as little as 30 minutes.

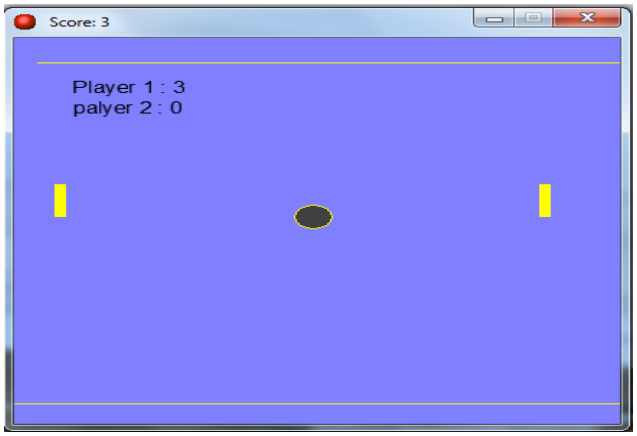

Figure 2. Prototype Pong Game designed using GameMaker

Designing the game in Game Maker requires very little or no programming knowledge. It has several in-built menu options that can be selected as per the user's requirements. Initially, we need to create the sprites or graphic images for the ball and the paddle. This is done using the built-in editor in minutes.

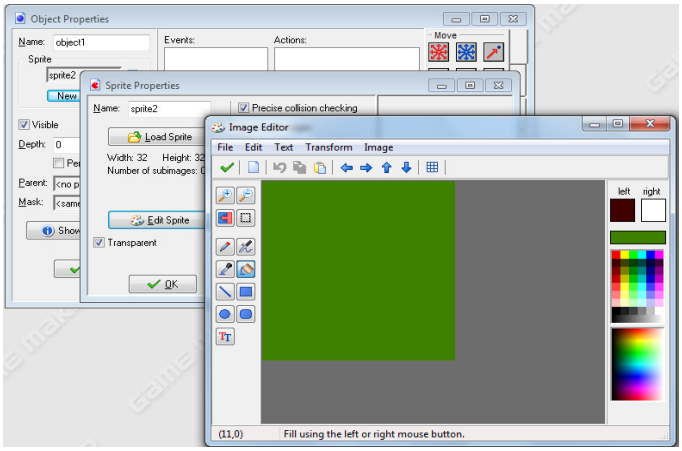

Figure 3. Designing an object

Once the required sprites are created, we use them as graphic images to represent the objects. After the objects are created, the next step is to specify events (such as collision between ball and paddle or key press by game player) and the actions performed by the objects in response to the events (e.g., bounce the ball back or move the paddle). This process is shown in Figures 3 and 4. With the use of events and actions, the game creator can iterate through different 'if-then-else' scenarios as required for the game.

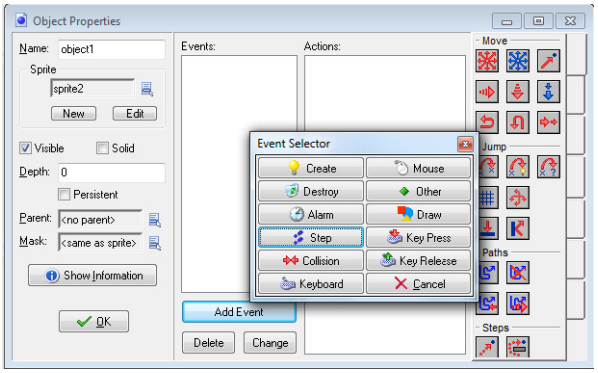

Figure 4. Adding events to objects

After assigning the required events and setting up other aspects of the game such as the score board and the room design where the objects are to be placed (see Figure 4), the final game is ready to be played.

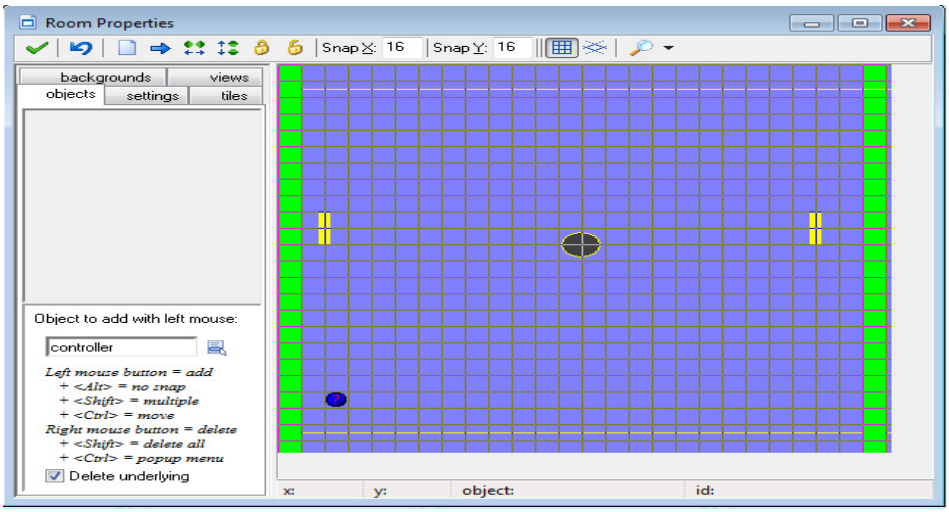

Figure 5. Design of room where the objects are to be placed

Theoretical Basis of Rapid Digital Game Creation-Based Learning

RDGC-based learning is grounded in the learning theories of social constructivism (Solomon 1994) and constructionism (Harel & Papert 1991). Piaget's theory of constructivism argues that knowledge and meaning are constructed rather than pre-existing (Piaget & Inhelder 1969). Experiences drive the development of ideas in a continuum that the learner ultimately derives meaning from. This makes the student a "builder" of knowledge, as opposed to a simple recipient of knowledge. This is at odds with the traditional classroom where a student is a quiet receptor and the teacher is a guardian of secret knowledge that is "gifted" onto the student. Social constructivism and constructionism (Harel & Papert 1991) go beyond constructivism by asserting that the best context for learning happens "when the learner is engaged in the construction of something external or at least shareable... a sand castle, a machine, a computer program, a book. This leads us to a model using a cycle of internalization of what is outside, then externalization of what is inside and so on." This is also a kind of "learning-by-making", as articulated by Papert (1991).

Constructionism is thus an epistemological framework concerned with building things, both in the sense of building understanding (as in constructivism) and building artifacts. RDGC is a constructionist learning activity because it involves the creation of a tangible artifact – a game, which in turn involves the designing of characters, virtual locations, and interactions of characters. Hence, it is a multi-layered constructionist process where building each artifact within the game is considered a separate, measurable instance of constructionist learning (Dalal et al., 2009). Basing on constructionism, learning in the RDGC environment happens from the process of creating the game and its components, experimenting with them to see how they work, modifying them to work better, and reflecting upon this process.

RDGC not only provides an opportunity to learn as a consequence of designing a game, it also allows for learning from the environment in which the game is developed, including fellow students and programming partners. Moreover, the learning is implicit, and a consequence of the main activity of designing a game. This is important, since game design is an exciting activity that provides a narrative to motivate students to continue engaging in the activity, without necessarily focusing on what they are learning. They can later reflect on what they learned, and thereby complete the loop for metacognition.

RDGC and Object-Oriented Concepts

There is some evidence that using a rapid prototyping tool in a classroom and lab before exposing students to formal programming would create a better understanding of O-O concepts and improve their programming skills (see e.g., Cooper, Dann, & Pausch, 2003). Scratch, a creation of MIT's media lab, has been used prior to teaching Java in an introductory computer science course at Harvard (Malan & Leitner, 2007) and it was found that the use of the software was exciting to students at a critical time during their first exposure to computer science and it helped the novice learners of programming to learn without the distraction of syntax.

Object-oriented thinking and its realization in the form of a prototype game is intrinsic to the RDGC process. Object orientation involves an intuitive if not explicit understanding of concepts such as objects, instances, events, abstraction, polymorphism, encapsulation among others. At a lower level, it also involves the understanding of programming structures such as sequence, decisions, and iterations. Table 1 shows some common O-O concepts also used in OOP languages such as Java. This is not meant to be a comprehensive list.

Table 1: Commong O-O Concepts

Concept	Description
1) Abstraction	Abstraction is used to represent essential characteristics without necessarily explaining all the details. This includes the notions of object classes and instances.
2) Inheritance	Inheritance allows an object of a class to acquire the properties of the object of a super class. This allows for reusability.
3) Polymorphism	Polymorphism allows an operation to take more than one form and hence can allow an object to show different behaviors in different situations.
4) Encapsulation	Encapsulation compartmentalizes the functional details of some or all of the object's components such that the internal details are hidden from view outside the object. This involves the concepts of properties and methods.

RDGC for O-O Learning And Teaching

How does designing a game such as Pong using an RDGC tool help the learner understand O-O concepts and how can an instructor demonstrate the concepts via the game? One teaching approach is to illustrate the concepts by having the students build a game and explore the concepts by means of their RDGC implementation, and then reflect on their exploration and experiences. This may be done in an introductory computing course or a pre-OOP course. This approach is consistent with an objects-first strategy recommended by curriculum experts of computing, where students learn O-O concepts first in contrast to the more commonly used programming-first approach (Cooper, Dann, & Pausch, 2003; Topi, Valacich, Kaiser, Nunamaker, Sipior, de Vreede, & Wright (2010).

The theoretical framework of constructionism suggests the following pedagogic guidelines:

1. Create different learning activities to be related to a larger task. This allows students to see the interconnectedness of different concepts and skills. With RDGC, the overarching goal of creating a prototype digital game will be the central theme for the learning activities, which include O-O learning.
2. The learner needs to be given ownership of the overall problem, which allows for free exploration of alternative solutions. Hence, Pong can be a starting point but learners should be allowed to make their own games.
3. An authentic task should be designed for the learner- i.e. the learner should feel that the game that they are creating will be fun to play, and will be used by others.
4. Allow reflection on the content being learned. Students may be asked to write a report for the game that they created along with a separate reflection essay on their experiences with RDGC and their understanding of O-O concepts. After creating several games, students may be asked to select one or two games to include in a digital portfolio, justifying why they picked those specific ones.
5. Later, when students learn an OOP such as Java, they can be asked to create an equivalent Pong game and asked to show the correspondences from the RDGC implementation to the equivalent Java code.

Hence, in an attempt to build a digital game, learners intrinsically learn basic OOP concepts without necessarily realizing that they are using those concepts. Subsequently, when they do learn an OOP language, it is easier for them to understand the programming constructs because they can be correlated with specific examples from the user's own game products. RDGC can also help better understand the basic concepts of programming such as the use of sequence, loops, decision structures, and other aspects of programming because they are direct implementations of earlier-performed intuitive RDGC tasks.

We discuss below the RDGC mappings of some representative O-O concepts.

Abstraction

In object-oriented programming, we are able to create abstract object classes and their instances, and specify properties and methods or operations. In the RDGC implementation of Pong, the learner can be shown e.g., that Paddle is an object class with properties of width and length and methods relating to movement direction and movement velocity. The specific paddles used in the game are instances of the class Paddle, a fact that would be intuitively obvious to the learner but can be explicated as a labeled concept by the instructor.

Inheritance

Inheritance allows classes to *inherit* commonly used state and behavior from other classes. For example, in the Pong game, the general Paddle object class can be shown to be used to create specialized classes representing various kinds of paddles such as LongPaddle, MediumPaddle, ShortPaddle, and each subclass would inherit the properties and methods from the superclass Paddle.

Polymorphism

Polymorphism as an O-O concept allows an operation to take more than one form and hence can allow an object to show different behaviors in different situations. As a programming language concept, polymorphism allows values of different data types to be handled using a uniform interface. For example, in the Pong game, the Move method for a Paddle or a Ball can be shown to be implemented in different ways depending upon the level and complexity of the game.

Encapsulation

Encapsulation is a language mechanism for restricting access to some of the object's components in an OOP. In the Pong game, encapsulation can be explained e.g., in terms of how the methods of an object are hidden from other objects. For example, the movement of a Paddle instance is not known to the Ball instance.

Discussion and Future Research

The use of game design in the curriculum is not new. Many studies have reported largely positive effects of game design in terms of attitudes, learning, creativity, holding the student's interest, retention, and other parameters and the empirical evidence for this approach is growing. In this paper, we have examined the use of RDGC in learning Object-Oriented concepts. We believe the learning process using RDGC facilitates the learning of the abstract concepts of Object-Oriented programming and modeling prior to actually programming in an OOP. While we have presented some evidence, our study is limited by its exploratory nature.

We do not view RDGC as an alternative to teaching a formal programming language such as Java. But we do see it providing advantage in the understanding of programming constructs if the instructor explains the constructs in terms of the steps taken by the student in the creation of his game. Therefore, we see RDGC and other formal programming languages complement one another. It can help flatten the steep learning curve needed to learn O-O computer programming. Moreover, based on our experience, RDGC can also be used in systems courses for object-modeling purposes as the game characters and props can serve as virtual-world objects for modeling.

We believe RDGC is a general pedagogical approach with wide ranging applications though there is a need for studies to test this assertion in different domains. It can also be used to teach domain-specific knowledge when students build games in specific domains. Recent studies (Lenhart et al., 2008) on the social impact of games show that electronic games cut across societal boundaries and provide a framework that is understood by diverse groups. Learning outcomes improve when students have a context for learning that is framed within their own experience (Bransford, Brown, & Cocking, 2000). The target audience of high-school and college-age students identifies with, and is strongly rooted in, the culture of games. Seymour and Hewitt's classic study (Seymour & Hewitt 1994) highlighted the adverse impact unfamiliar college culture has on retention and success in STEM programs. Since students from all backgrounds understand games, they

have broad and deep experience to draw from in game creation, contextualizing design. Since the appeal of games transcends gender, age, and race, introducing RDGC can potentially increase computing enrollment among groups historically known to be under-represented in those disciplines.

Games encourage interaction and community building, even between individuals with different levels of expertise. Support networks spring up rapidly around popular games through on-line forums and other social networks. (Gee, 2004). These forums serve as effective informal learning environments that allow players of different ages and experience to rapidly become more proficient. Games thus spontaneously form structures similar to learning communities, which have significant positive impact on retention (Hotchkiss, Moore, & Pitts, 2006; Tinto, 1998).

Learning O-O as a facet of learning computational thinking has implications not only for skills in computing but also for other fields of study and in general problem-solving, although this area requires further research.

Our exploration raises several research issues for pedagogy related to RDGC. Although there is some documented evidence, there is need for rigorous empirical studies to understand how good the learning of O-O concepts is when this approach is used. There is also a need for effective pedagogic models and best practices for the use of this approach in the classroom. Other issues that emerge include research into the use of pre-built template games for imparting domain-specific knowledge and O-O skills. We need to explore the types of games that appeal to different kinds of users in order to facilitate the building of an effective RDGC pedagogic framework.

Conclusion

A large body of research from multiple fields demonstrates the power of digital games in learning. In this paper, we have focused on learning that occurs from making rapid games using rapid game generation software. Computing education needs more innovative ways of instruction. We believe that rapid digital game creation has the potential to be an effective pedagogical model in IS and computing courses. Based on our exploration, RDGC holds promise as an important part of what may be called a "games first" approach to introductory programming (Leutenegger & Edgington, 2007). We have cited some empirical work done in this field but there is a need for more systematic studies of the relationships between different aspects of learning and RDGC-based pedagogies.

RDGC is also a useful pedagogic tool for other academic areas and not just content areas that require computer programming. Game construction and game playing provides more flexibility since it uses a variety of objects and scenarios in an interactive environment. Curriculum designers must consider the inclusion of RDGC in a variety of courses as O-O thinking has value outside of computing. Providing students with pre-designed games templates and guiding them to build computer games rapidly constitutes a creative approach for increasing interest in the computing disciplines.

Acknowledgements

We thank Praveen Kuruvada and Daniel Asamoah for their assistance during an earlier phase of this work.

References

Babcock, P., & Marks, M. (2010). *Leisure College USA: The decline in student study time.* American Enterprise Institute.

Bayliss, J. D., & Strout, S. (2006). Games as a "Flavor" of CS1. *SIGCSE'06*, Houston, Texas, USA.

Bransford, J. D., Brown. A. L., & Cocking, R. R. (2000). *How people learn: Brain, mind, experience, and school*. Washington, DC: National Academy Press.

Conway, M. J. (1997). *Alice: Easy-to-learn 3D scripting for novices.* University of Virginia.

Cooper, S., Dann, W., & Pausch, R. (2003) Teaching objects first in introductory computer science. *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, Reno, NV, February, 191-195.

Dalal, N., Dalal, P., Kak, S., Antonenko, P., & Stansberry, S. (2009). Rapid digital game creation for broadening participation in computing and fostering crucial thinking skills. *International Journal of Social and Humanistic Computing*, *1*(2), 123-136.

Davies, S. (2008). *The effects of emphasizing computational thinking in an introductory programming course*. Saratoga Springs, NY.

Ferdig, R., & Boyer, J. (2007). Can game development impact academic achievement? *T.H.E. Journal. [Online]*. Available: http://www.thejournal.com/articles/21483

Gee, J. (2004). *Situated language and learning*. New York, Routledge.

Gee, J. (2007). *What video games have to teach us about learning and literacy.* New York: Palgrave Macmillan.

Guzdial, M. (2008). Paving the way for computational thinking. *Communications of the ACM*, *51*(8), 27.

Habgood, L., & Overmars, M. (2006). *The game maker's apprentice: Game development for beginners.* Berkeley, CA: Apress.

Harel, I. E., & Papert, S. E. (1991). *Constructionism*. Ablex Publishing.

Hotchkiss, J. L., R. E. Moore, & Pitts,. M. (2006). Freshman learning communities, college performance, and retention. *Education Economics. 14*(2), 197-210.

Lenhart, A., Kahne, J., Middaugh, E., Macgill, A., Evans, C., & Vitak, J. (2008). *Teens, video games, and civics.* Washington, D.C.: Pew Internet & American Life Project.

Leutenegger, S., & Edgington, J. (2007). A games first approach to teaching introductory programming. *SIGCSE '07: Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, pp. 115–118, New York, NY, USA, 2007. ACM Press

Lu, J., & Fletcher, G. (2009). Thinking about computational thinking. *SIGCSE' 09.*

Malan, D. J., & Leitner, H. H. (2007). Scratch for budding computer scientists. *SIGCSE Bulletin, 39*(1), 223-227.

Moreno-Ger, P., Burgos, D., Martinez-Ortiz, I., Sierra, J. L., & Fernandez-Manjon, B. (2008). Educational game design for online education. *Computers in Human Behavior, 24*(6), 2530-2540.

Parberry, I., Kazemzadeh, M. B., & Roden, T. (2006). The art and science of game programming. *SIGCSE'06*, Houston, Texas, USA.

Papanastasiou, E. C., & Ferdig, R. E. (2006). Computer use and mathematical literacy: An analysis of existing and potential relationships. *Journal of Computers in Mathematics and Science Teaching*, 25(4), 361-371.

Piaget, J., & Inhelder, B. (1969). *The psychology of the child.* New York: Basic Books.

Seymour, E., & Hewitt, N. (1994). *Talking about leaving: Factors contributing to high attrition rates among science, mathematics, and engineering undergraduate majors.* Boulder, CO: Bureau of Sociological Research, University of Colorado.

Silveira, I., Araújo, C., Veiga, J., Naito, L., & Comotti, L. (2011). Building computer games as effective learning tools for digital natives – and similars. *Issues in Informing Science and Information Technology*, *8*, 77-92.

Solomon, J. (1994). The rise and fall of constructivism. *Studies in Science Education*, 23, 1-19.

Tinto, V. (1998). Colleges as communities: Taking research on student persistence seriously. *Review of Higher Education 21*(2), 167-177.

Topi, H., Valacich, J. S., Kaiser, K., Nunamaker, Jr., J. F., Sipior, J. C., de Vreede, G. J., & Wright, R. T. (2010). IS 2010: Curriculum guidelines for undergraduate degree programs in information systems. *Communications of the Association for Information Systems, 26. Article 18.* Available at: http://aisel.aisnet.org/cais/vol26/iss1/18

Wing, J. (2006). Computational thinking. *Communications of the ACM*, *49*(3).

Wright, J. (2007). *Thinking object-oriented*. Retrieved November 20, 2011 from http://jacwright.com/19/thinking-object-oriented

Biographies

Nikunj Dalal is Professor of Management Science and Information Systems in the Spears School of Business at Oklahoma State University in Stillwater. His research interests include learning, wisdom computing, rapid computer game creation, philosophical issues in information systems, modeling, and Web perception.

Subhash Kak is Regents Professor and Head of Computer Science Department at Oklahoma State University in Stillwater. His technical research is in the fields of information theory, neural networks, and quantum information. He has also written on history of science and on art. Amongst his awards include British Council Fellow (1976), Science Academy Medal of the Indian National Science Academy (1977), Kothari Prize (1977), UNESCO Tokten Award (1986), Goyal Prize (1998), National Fellow of the Indian Institute of Advanced Study (2001), and Distinguished Alumnus of IIT Delhi (2002).

Sohum Sohoni, is an Assistant professor in the School of Electrical and Computer Engineering at Oklahoma State University. His research expertise is in engineering education and computer engineering. His most recent work in engineering education is in the research and development of a learning platform that enables students to connect concepts they learn in different courses through a material anchor. In computer engineering research, he is looking at security issues in cloud computing. He received the CEAT Halliburton Excellent Young Teacher Award (2009) and the Regents Distinguished Teaching Award (2010).