Je-LKS
PEER REVIEWED PAPERS
Vol. 12, n.2, 2016
ISSN: 1826-6223 | eISSN: 1971-8829
Journal of e-Learning and Knowledge Society
The Italian e-Learning Association Journal

USING SERIOUS GAME TO ENHANCE ALGORITHMIC LEARNING AND TEACHING

Wassila Debabi
Tahar Bensebaa

Computer Science Department, Laboratory of Research in Computer Science (LRI), Badji Mokhtar-Annaba University, Annaba, Algeria - wassiladebabi@gmail.com; tahar.bensebaa@univ-annaba.org

Keywords: Algorithmic learning, Serious game, Learning games, Education.

Data structures and algorithms are important foundation topics in computer science education. However, they are considered to be hard to teach and learn because usually model complicated concepts, refer to abstract mathematical notions, or describe complex dynamic changes in data structures. Many students in programming courses have difficulties to master all required competencies and skills especially at introductory level. In the literature there are different ways to enhance learning programming and deal with the important dropout rate. Recently, games are increasingly being used for education in various fields. We hypothesize that games have the potential to be an important teaching tool for their interactive, engaging and immersive activities. So they can improve student engagement, motivation, and consequently learning. To this end, we are developing a game to teach basic algorithmic concepts and algorithms. We aim to initially investigate the educational games developed for and used in the computer programming

for citations:
Debabi W., Bensebaa T. (2016), *Using Serious Game to enhance algorithmic learning and teaching,* Journal of e-Learning and Knowledge Society, v.12, n.2, 127-140. ISSN: 1826-6223, e-ISSN:1971-8829

domain and review to which level they address the aforementioned difficulties. Then, we propose a role playing game called *AlgoGame* based on existing solutions and incorporates new elements. A pre validation of the game with novice students was very encouraging and demonstrates that learning programming can be enhanced by playing with *AlgoGame*.

1 Introduction

Learning programming and its basics are fundamental skills that all computer science students are required to learn during their curricula. For beginners, this learning can be in many cases difficult and challenging, especially if it is their first contact with this area. The specificities of those difficulties and the corresponding causes continue to be an ongoing topic of research, many reasons are pointed out for this, and several researches has been carried out to recognize the characteristics of novices programmers. Mostly, the traditional learning methodologies usually based on doing countless exercises that aim to cover many areas, but are often disconnected from each other and can become tiresome, as they offer little immediate rewards to the student (Coelho *et al.*, 2011) Also, the programming languages typically used in programming classes are professional in nature (C, C++, C# and Java) they have extensive and complex syntaxes, rendering learning difficult for beginners (Jenkins, 2002; Motil & Epstein, 2000).

Nowadays, technology has become crucial in educational development and for the revolution in learning systems (Olapiriyakul *et al.*, 2006). Technology creates and transforms the learning and teaching processes, which brings new opportunities to the educational system (Esteves *et al.*, 2011).

Serious games technology offers tools that may have potential to help computer programming students to become more engaged on their learning through a `learn while having fun' approach (Coelho *et al.*, *op.cit.*). We propose in this paper a new game were learning how to program can be enhanced and encouraged through this type of approach, by creating a serious game that is both a conduit of knowledge and experience and at the same time a fun task.

The organization of the paper is as follows. Initially, we expose our theoretical framework in Section 2, and then, we present a brief study and discussion about the existing educational games and introduce our proposed game in Section 3. Section 4 includes our methodological framework, and a study case is presented in Section 5. Our experimental results and their implications are related in Section 6. Finally, conclusions drawn from the work done so far are provided in Section 7.

2 Theoretical framework

Programming is known for its complexity and difficulty. It is believed to

be hard to teach and to learn and many students in programming courses have difficulties to master all required competencies and skills. This learning represents a challenging task especially at introductory level and often has the highest dropout rates (Robins *et al.*, 2003). Many reasons are pointed out for this, mainly, the traditional teaching methods are generally based on lectures and specific programming language syntaxes, rendering learning difficult for beginners, and fail often to attract and motivate them so they get implicated in programming activities.(Lahtinen *et al.*, 2005; Schulte & Bennedsen, 2006). Another concern is the students' difficulties with abstract concepts: knowing how to design a solution to a problem, subdivide it into simpler code able subcomponents, and conceive hypothetical error situations for testing and finding out mistakes (Esteves *et al.*, 2008); and difficulties in understanding even the most basic concepts (Lahtinen *et al.*, *op. cit.*; Miliszewska & Tan, 2007) such as variables, data types or memory addresses as these abstract concepts do not have direct analogies in real life (Lahtinen *et al.*, *op. cit.*; Miliszewska & Tan, *op. cit.*); Actually, the students apply the human principles of thinking and acting to the computer. We may therefore say that the first step of learning programming is going from human thinking to programming concepts. This includes understanding the way programs are executed, both in terms of internal variables and external files and I/O (Kaasboll, 1998). Many technological solutions attempt to solve these problems by providing environments who give facilities in tasks of execution and visualization mechanisms, while they failed in other aspects such as the lack of motivation and the inability to properly portray in a comprehensible way the complex computer programming concepts (Henriksen & Kölling, 2004). To solve these issues, a new wave of educational environments emerge, called "MicroWorlds" , like StarLogo The Next Generation (Klopfer & Yoon, 2004); Scratch (Maloney *et al.*, 2004); Alice (Kelleher *et al.*, 2002) that include algorithm and data structure visualizations and animations in order to improve their learning (Shaffer *et al.*, 2010). The dynamic and symbolic images in an algorithm animation help to provide a concrete appearance to the abstract notions of algorithmic methodologies, thus making them more explicit and clear (Kehoe *et al.*, 2001). Although these environments provide better visualization of concepts feedback but students, especially novices continue to face significant problems in computer programming courses (Lahtinen *et al.*, *op. cit.*; Ragonis & Ben-Ari, 2005). The lack of motivation and facilitating methods that guide students through understanding complex concepts are only two of the most evident problems that have yet to be successfully overcome and thus prevent successful computer programming education (Pears *et al.*, 2007).

Subsequently, heads are being turn toward other technologies that can make students more interested and facilitate the task for the teachers. To this end,

educational games have started gaining teachers' attention and used as a motivator for students in computer curricula and research (Eagle & Barnes, 2009). Serious games and games for learning have recently been suggested as an engaging way of helping students to learn (JISC, 2007). Interest in serious games emerged initially from speculation that games could provide highly engaging activities which could be utilized in learning (Boyle *et al.*, 2011). More importantly games offer methods of learning that are highly consistent with modern theories of effective learning which propose that learning activities should be active, situated, problem-based, interactive and socially mediated (Boyle *et al.*, *op. cit.*).

As previously said, acquiring and developing knowledge about programming is a highly complex process, and it involves a variety of cognitive activities, and mental representations related to program design, program understanding, modifying, debugging (and documenting). (Rogalski & Samurçay, 1990, p. 170). Our principal attention carries on both program design and program understanding process. Where our goal was articulated according three aspects. The first one is to let students focus on the resolution of the problem more than the syntax of programming language. The second is to reduce their difficulties with abstract concepts. And the last, is to allow them knowing how to design a solution to a problem by dividing it into simpler problems.

Thus, we hypothesize that teaching algorithmic concepts through playing a game can improve student engagement, motivation, and consequently learning because these factors, according to researchers (Garris, *et al.*, 2002), may be the most important factors in learning. To this end we propose a new game called *AlgoGame* to enhance algorithmic learning and teaching.

3 Related works and proposed game

In this section, we present the most known works in the field of educational games and particularly games dedicated to learning programming. For each game, we briefly describe the targeted concepts, scenario and features.

GAME2LEARN (Barnes *et al.*, 2008) this project groups two games with two distinct scenarios. "Saving Sera" is a 2D exploratory game, implemented using RPGMaker. The user must perform various tasks involving programming concepts: correctly reordering a while loop statement of a confused old fisherman's mind; correcting a nested for loop placing eggs in crates; and visually piecing together a quicksort algorithm. When the player makes a mistake, the character must fight a script bug, which asks the users various computer science questions in order to fight the bug.

The second game is "The Catacombs" a 3D game developed using the Bio-

Ware Aurora toolset. In this game, the user is an apprentice wizard who must perform three progressively more complicated tasks. The first involves two if statements; the second, a nested for loops; and the third, solving a cryptogram using more nested for loops. In the second and third quests, incorrect answers resulted in decreasing player health.

Code Studio (Code Studio, 2015) it is the application connected to the well known "Hour of Code" initiative, an introduction to computer science, designed to demystify code and show that anybody can learn the basics using Scratch. This global movement reaches tens of millions of students in different countries. It is organized by Code.org. A coalition of partners have come together to support the Hour of Code too, including Microsoft, Apple, and Amazon.

Prog & Play (Muratet *et al.*, 2010), it is a real-time strategy game, where students write programs to control units in a battlefield. They can choose which programming language they want to use amongst Ada, C, Java, OCaml, Scratch and Compalgo. Prog&Play is a multiplayer game that allows students to interact with each other.

Wu's Castle (Eagle & Barnes, 2009) It is a 2D role playing game, developed in RPG Maker XP, where students program changes in loops and arrays in an interactive, visual way. The game provides immediate feedback and helps students visualize code execution in a safe environment. The player interacts with the game in two ways: by manipulating an array through changing the parameters within the loop or by moving the game's character (wizard) through executing nested loops. Wu's Castle use the programming language C ++.

CoLoBot: Colonize with Bots (CoLoBot, 2013) it combines both a real time 3D game of strategy and an initiation to programming. The player is at the head of a space expedition and he is assisted only by some robots. The mission consists in successive attempts at the exploration and colonization of various planets. The player will have to search for the raw materials and energy he needs in order to survive. CoLoBoT aims to teach students Object Oriented programming style similar to C++ or Java.

PlayLogo3D (Paliokas *et al.*, 2011), it is a 3D role playing game, especially designed for children aged 6-13 years in the early stages of programming education. The scenario portrays the rivalry between pilot robots (players) in the "spaceship X-15". Although this is a multiplayer game, voice or text communication between players is not allowed during the game, in order to allow better concentration for learners. PlayLogo3D aspires to introduce the very ba-

sic concepts of structured programming using LOGO programming language.

RoboCode (O'Kelly & Gibson, 2006) Developed by IBM and released in 2001, Robocode is a game (and also a development environment) that aspires to teach programming using Java language. During the game, the player tries to program a robot that will fight another robot in a virtual arena. The concepts taught by Robocode are the basic concepts of structured programming but also the main structures of object-oriented programming such as inheritance, polymorphism, etc.

M.U.P.P.E.T.S (Phelps *et al.*, 2003) "The Multi-User Programming Pedagogy for Enhancing Traditional Study" is a web-based, collaborative three-dimensional game that aims to teach the basic concepts of object-oriented programming using exclusively the JAVA programming language. Students are called to write and manipulate "objects" that can use them later in the game. MUPPETS is inspired by Robocode, thus players create robots that will fight in a virtual arena.

3.1 Proposed game

All the initiatives mentioned above propose a number of features that meet the problems typically encountered in computer programming. They introduce interesting notions such as attractive graphical interfaces, visual representation of the programming tasks, interactive and interesting scenarios that keep the learner immersed. We can distingue two categories of games, the first aims to teach a specific computer programming unit like Catacombs and Saving Sera from GAME2LEARN project, Prog & Play, Vu's Castle and ColoBot while the second category includes games that cover multiple pedagogical goals such as PlayLogo3D, RoboCode and MUPPETS. These games attempt to motivate learner by asking him to develop his own game, as in the case of Robocode and MUPPETS, or in another approach, to play a game and code to progress, in projects like GAME2LEARN, Prog & Play, etc.

Additionally, some games like Catacombs and Saving Sera use micro-languages to support students' understanding of the logic behind the programming elements, while others use a distinct programming language for teaching such as Pascal, JAVA, C / C ++. We believe that is a challenging situation for the learner. The algorithmic thinking is a key ability in informatics that can be developed independently from learning programming (Futschek, 2006) and dealing with the syntax of a particular programming language is an additional charge, especially for novice learners that may discourage them and lead them to give up learning. Besides, some games propose to learner filling the gaps or

answer Multiple Choice Questions (MCQ), in this case, the learner can answer randomly without truly learn nothing.

Our work is based on these tools; we are oriented towards the second approach, so we propose to teach algorithmic concepts through a serious game that allows the learner to focus more on solving the problem rather than on the syntax of the programming languages. Several common deficits in novices' understanding of specific programming language constructs were pointed out in many studies (Soloway & Spohrer, 1989; Jenkins, 2002) and inappropriate analogies may be drawn from natural language (Soloway *et al.*, *op. cit.*) leading learners to serious confusion. Another concern is about actions that take place "behind the scene": the abstraction is a powerful programming concept but beginners face difficulties moving from the abstract toward the concrete (Soloway *et al.*, *op. cit.*). *AlgoGame* gives the player a dynamic translation of his actions in the game into algorithmic instructions. Thus, students pass the river between the tangible world and the abstract world of programming environments and languages smoothly through the game.

AlgoGame is designed essentially for computer science students that are coming into contact for the first time with this area. It aims to introduce them to algorithmic concepts, to help them get acquainted with the way algorithms are structured and also allow them to engage in algorithmic thinking. The game has several levels; each one aspires to teach a basic algorithmic concept (alternative "if"; iteration "loops"; sequence of command, etc) or an algorithm that combines these concepts.

4 Methodological Framework

4.1 Constraints and specifications

The critical point of a serious game is the relationship between the game and its educational content. Several experiments have shown that serious games achieve their goals when they have a strong "game" component clearly highlighted. Thus, serious games have double scripting. Actually, there are two scenarios to design: a game scenario (explicit) in accordance with another learning scenario (implicit). Our idea is that the game has n-different levels; each level is dedicated (implicitly) to a learning goal. In each level, the game simulates the algorithm's behavior.

Another fundamental point is how to maintain the player's motivation during the game. Immersion is a key point for motivation and collaboration. There is also another aspect that must be considered. When students learn something in a game, they need to transfer the acquired knowledge to real life. If the game includes elements that match the real situation, this transfer will be simplified

(Carron *et al.*, 2009). The authors give some guideline in order to take into account the immersion feature as much as possible while designing the educational game (1) Develop a coherent world (2) Find the real places or people usually involved in the learning process that you would like to integrate into the game (3) Design an overall pedagogical scenario, a story, taking place in the premises defined in the game design (4) Find which rules of the real world you need to reproduce in the game (5) Define metaphors linked with the general story and with the different learning objects. Thus, the serious game we want to develop must meet the set of aforementioned constraints and specifications.

4.2 The Mechanics

Every game has a set of rules that indicates how it is meant to be played by the players (the game mechanics) serving as a basis for the gameplay; the serious game proposed in this paper is designed taking in consideration the constraints and the specifications reported in (4.1.) to meet the aimed academic objectives. This section will cover the core mechanics proposed for our game.

Proposed Core Mechanics: The main learning objective of the game is to lead player to generate a coherent algorithm or a sequence of command through his actions in the game. Each action made in the game generates a specific command. This section describes the core elements of the game mechanics that meet the aforementioned specifications.

- The game takes place in the computer science department. The main character is a "Byte" (caricatured) who goes in the exploration of locations after his escape from a student's Personal Computer.
- The game is designed as a series of levels connected to explorable interconnected rooms. Each room includes a mission for a given algorithmic concept.
- In every room, "Byte" discovers a mission (related to an algorithmic concept or an algorithm). The resolution of a mission opens new explorable areas, before forbidden to access (next mission).
- During a mission, hints or non-player characters may appear in order to guide the player, provide useful information, etc.
- While playing, the learner will visualize the transformation of his actions in the game to algorithmic commands in the right of the screen. (concretize abstract concepts)
- The final score is calculated from the scores made during previous missions. The player has three lives for each mission. If completed successfully, the full score is assigned, otherwise, a reduction of points from the score until all lives are consumed.
- The teacher can add other concepts (according to his learning objective)

and keep these same set of rules, just insert a new mission or redirect the learner to specific missions.

5 Case Study

This section will describe a case study. In fact, *AlgoGame* has several levels; each one related to a basic algorithmic concept (alternative "if"; iteration "loops"; sequence of command) or to an algorithm that combines these concepts. We choose to describe the level connected to the "selection sort algorithm", since it allows us to illustrate several algorithmic concepts.

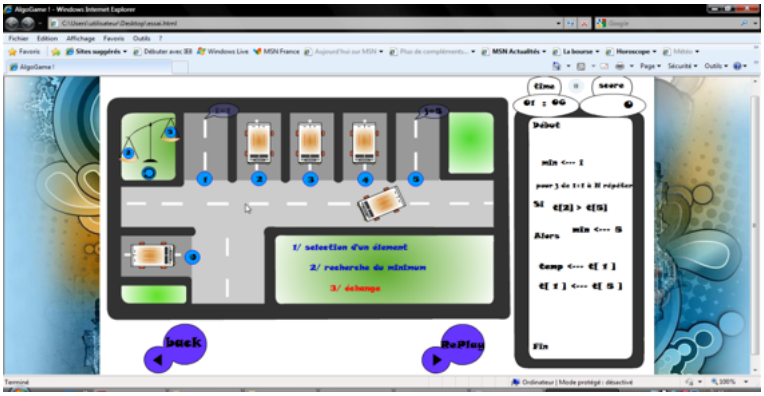

Fig. 1 - Third phase of the algorithm: Swap elements.

Gameplay. The player is situated in the Computer science department parking; he has to sort, within a given time, parked vehicles in garages. This sorting will be done according to vehicle weight, from the least heavy to the heaviest. To achieve this, the learner must complete a series of specific tasks to establish the desired order before time limit; otherwise, the game is over.

The selection sort algorithm consists in sorting array elements in a given order. In order to concretize the abstract concepts related to this algorithm, we thought to materialize the array cells by garages and items by parked vehicles within these garages.

Furthermore, the vehicle weight is not communicate to the player, we suggest to indicate (graphically) only, the number of the least heavy truck, so the player has no choice other than browse all the vehicles to determine the minimum, as this actually happens in the algorithm.

To swap two vehicles, the player must go through an intermediate zone because the road leading to the various garages passes one vehicle at a time. This operation allows the learner to get aware of the "machine" constraints, i.e.

the need to use temporary variable when swapping two elements.

Every action made in the game generates an algorithmic command which appears at the right of the screen, until the whole algorithm is constructed.

The game includes explanatory messages that assist students in understanding the theory while playing, such as the hints with indexes of the array. Besides, the selection sort algorithm goes by three phases (1) Select an element (2) Find the minimum or the maximum (3) Swap the tow elements. In order to highlight these phases, the player is informed each time he goes through one of them.

6 Experimental Results

This experiment took place with first year students learning computer science at the Preparatory School in Sciences and Techniques, Annaba. We chose 33 volunteers students. Students were novice at the time of the experiment; they did not know any sorting algorithm.

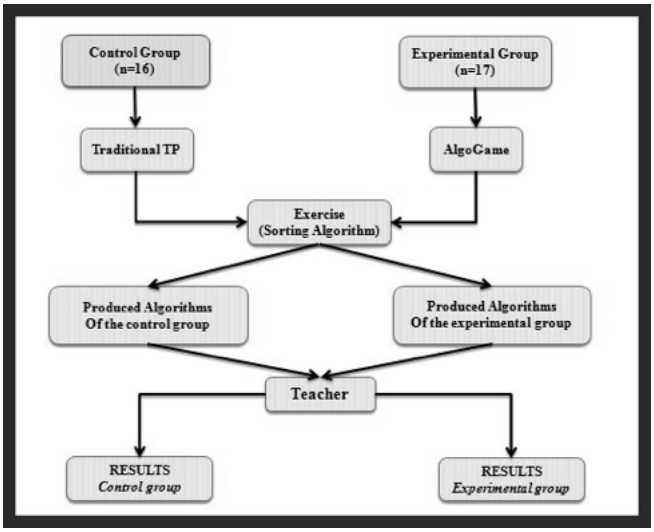

Fig. 2 - Experimental design

The control group of students (n=16) were asked to write an algorithm of sorting without knowing about the game, only an explanation of the algorithm's principle was given.

The experimental group (n=17) were asked to proceed as following:

- Playing the game (*AlgoGame*) for 30 minutes,
- Write the sorting algorithm.

Table 1
DESCRIPTIVE STATISTICS

	Control group	Experimental group
Mean	0.45	1.13
SD	0.92	1.28
median	0.00	0.88
Max	3.75	4.00
Min	0.00	0.00

As previously said, both experimental and control groups were asked to write a sort algorithm in the end of the session, the figure below shows the score made by each group after correction of the produced algorithms by the teacher.

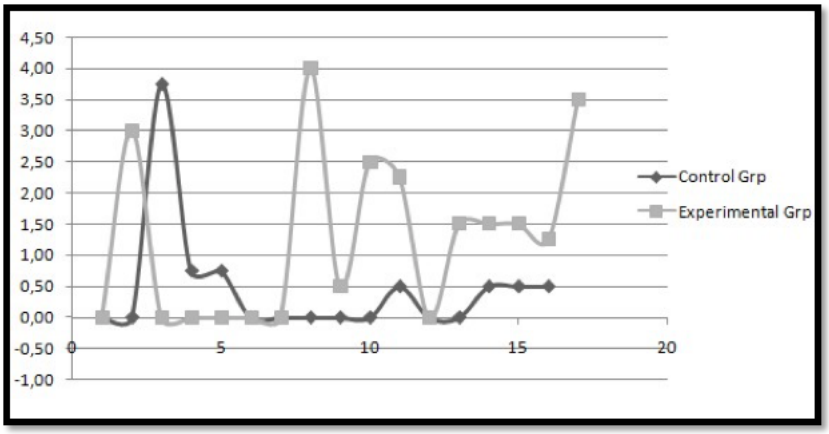

Fig. 3 - Score Control group VS Experimental group

We notice that the experimental group has a better Mean comparing to the control group (Table 1, Figure 3). Even if with a small experimental group, our results were encouraging and positive and show that students who played *AlgoGame* produced a better resolution of the problem. Thus, we aim to extend this experience to a larger population in order to discuss and validate the learning outcomes.

Conclusions and future work

AlgoGame is a novel way for learning and teaching algorithmic bases that has been presented in this paper. Our principal concern was to reduce the ab-

straction and help students to make relation between their real world and the abstract algorithmic concepts. Also, it allows them to realize that a problem can be divided into simpler sub problems by highlighting the algorithm's steps when the player goes through them. The game discharge student from syntax and debug errors, thus the algorithms are not constructed by typing text or arranging icons but by taking actions in the game.

At this moment, a game prototype was implemented but teacher interface and features are in progress. This one will allow the teacher adding new concepts or modify the existing levels' sequence according to his learning objective.

In the next step, we aim to incorporate our game into an Adaptive Online Distance Learning platform to insure the adaptation of the learning scenario (and consequently the game scenario) according to learners needs. Actually, the diversity of learners, this phenomenon found in all classrooms fact that each learner is characterized by some diversity in the results, abilities, interests, motivation and needs. Our goal is the contextualization and the individualization of the gaming experience and therefore learning experience for each learner according to his own needs.

REFERENCES

Barnes, T., Chaffin, A., Powell, E., Lipford, H. (2008), *Game2Learn: Improving the motivation of CS1 students*, in: Proceedings of 3rd international conference on Game development in computer science education.1-5.

Boyle, E., Connolly, T. M., & Hainey, T. (2011), *The role of psychology in understanding the impact of computer games*. Entertainment Computing, 2(2), 69-74.

Coelho, A., Kato, E., Xavier, J., & Gonçalves, R. (2011), *Serious game for introductory programming*. In Serious Games Development and Applications(pp. 61-71). Springer Berlin Heidelberg.

Code Studio: https://studio.code.org/. Accessed 28 August 2015.

CoLoBoT : http ://www.ceebot.com/colobot/index-e.php. Accessed 18 September 2013.

Eagle, M., Barnes, T.(2009), *Experimental evaluation of an educational game for improved learning in introductory computing*. ACM SIGCSE Bulletin 41(1), 321-325.

Esteves, M., Fonseca, B., Morgado, L. & Martins, P. (2008), *Contextualization of programming learning: a virtual environment study*, in: Proceedings of 38th ASEE/ IEEE Frontiers in Education Conference. 17–22,Washington, DC: IEEE.

Esteves, M., Fonseca, B., Morgado, L., & Martins, P. (2011), *Improving teaching and learning of computer programming through the use of the Second Life virtual world*. British Journal of Educational Technology, 42(4), 624-637.

Futschek, G. (2006), *Algorithmic Thinking: The Key for Understanding Computer*

Science. Springer-Verlag Berlin Heidelberg 2006, pp. 159–168.

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

Henriksen, P., Kölling, M. (2004), *Greenfoot: Combining Object Visualization with Interaction*, in: Proceedings of 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications. 73-82.

Jenkins, T. (2002), *On the difficulty of learning to program*, in: Loughborough University, UK, 3rd Annual LTSN_ICS Conference. 53–58, The Higher Education Academy.

JISC: Joint Information Systems Committee. (2007), *Game-based learning*: briefing paper (online): http://www.jisc.ac.uk/media/documents/publications/gamingreportbp.pdf

Kaasbøll, J. J. (1998), *Exploring didactic models for programming*. In NIK 98– Norwegian Computer Science Conference. p.p. 195-203.

Kehoe, C., Stasko, J. & Taylor, A. (2001), *Rethinking the evaluation of algorithm animations as learning aids: an observational study*. Journal of Human-Computer Studies (2001) 54, p.p. 265-284

Kelleher, C., Cosgrove, D., Culyba, D., Forlines, C., Pratt, J. & Pausch, R. (2002), *Alice2: Programming without syntax errors*, in: Proceedings of 15th annual symposium on the User Interface Software and Technology.

Klopfer, E., Yoon, S. (2004), *Developing games and simulations for today and tomorrow's tech savvy youth*. TechTrends, 49(3), 33–41.

Lahtinen, E., Mutka, K. A. & Jarvinen, H. M. (2005), *A study of the difficulties of novice programmers*, in: Proceedings of 10th Annual Conference on Innovation and Technology in Computer Science Education. 14–18, NewYork: ACM Press.

Maloney, J., Burd, L., Kafai, Y., Rusk, N. Silverman, B. & Resnick, M. (2004), *Scratch: A sneak preview*, in: Proceedings of 2nd International Conference on Creating Connecting, and Collaborating through Computing. 104–109. IEEE Computer Society.

Miliszewska, I. & Tan, G. (2007), *Befriending computer programming: a proposed approach to teaching introductory programming*. Journal of Issues in Informing Science & Information Technology, 4, 277–289.

Motil, J. & Epstein, D. (2000), *Jr: a language designed for beginners (less is more)*. Retrieved July 10, 2014, from http://www.csun.edu/~jmotil/BeginLanguageJr.pdf

Muratet, M., Torguet, P., Viallet, F & Jessel, J.P. (2010), *Experimental feedback on Prog&Play, a serious game for programming practice*, Eurographics, 1-8.

O'Kelly, J. & Gibson, J. P. (2006), *RoboCode & problem-based learning: a non-prescriptive approach to teaching programming*, in: Proceedings of Conference on Innovation and Technology in Computer Science Education. 217–221. NY: ACM.

Olapiriyakul, K. & Scher, J. M. (2006), *A guide to establishing hybrid learning courses: employing information technology to create a new learning experience, and a case study*. The Internet and Higher Education, 9, 287–301.

Paliokas, I., Arapidis, C. & Mpimpitsos, M. (2011), *PlayLOGO 3D: A 3D interactive*

video game for early programming education, in: Proceedings of 3rd International Conference on Games and Virtual Worlds for Serious Applications. 24-31.

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

Phelps, A.M. Bierre, K.J. & Parks, D.M. (2003), *MUPPETS: multi-user programming pedagogy for enhancing traditional study*, in: Proceedings of 4th conference on Information technology education. 100-105.

Ragonis, N., Ben-Ari, M. (2005), *A long-Term Investigation of the Comprehension by Novices*, Computer Science Education, 15(3), 203-221.

Robins, A., Rountree, J. & Rountreen, N. (2003), *Learning and teaching programming: a review and discussion*. Computer Science Education, 13(2), 137–172.

Rogalski, J., & Samurc¸ ay, R. (1990), *Acquisition of programming knowledge and skills*. In: J.M. Hoc, T.R.G. Green, R. Samurc¸ ay, & D.J. Gillmore (Eds.),Psychology of programming (pp. 157 –174). London: Academic Press.

Shaffer, C. A., Cooper, M. L., Alon, A. J. D., Akbar, M., Stewart, M., Ponce, S., & Edwards, S. H. (2010), *Algorithm visualization: The state of the field*. ACM Transactions on Computing Education (TOCE), 10(3), 9.

Schulte, C. & Bennedsen, J. (2006), *What do teachers teach in introductory programming?* In: Canterbury, UK, 2ed International Workshop on Computing Education Research. 17–28, New York: ACM.

Soloway, E. & Spohrer, J. (1989), *Studying the Novice Programmer*, Lawrence Erlbaum Associates, Hillsdale, New Jersey. p.497.