Code Factory – A game environment to start programming

Anabela Gomes, Pedro Belchior
Nuno Martins, César Páris, Álvaro Santos
ISEC – Engineering Institute of Coimbra – Polytechnic Institute of Coimbra
Portugal
anabela@isec.pt, a21180321@alunos.isec.pt
ncmartin@isec.pt, cparis@isec.pt, ans@isec.pt

Abstract: There is a high worldwide failure rate in the first programming subjects. There are various causes such as the abstraction capabilities, the problem solving skills or the math's background required for learning programming that can contribute to this phenomenon. Another aspect is that, when first being introduced to programming subjects, many students feel that their learning experience is hindered by a lack of motivation. We find that this may stem from the static nature of the teaching methods used in introductory programming courses. The lack of simple tools, where novice students can apply their newly acquired knowledge, without being distracted by overly complex user interfaces and technical feedback is another problem. The environment presented in this document aims to provide students with an interactive game that helps them consolidate core-programming skills. By letting students observe and explore a game environment while being guided through a predefined set of puzzle-like exercises, we hope to create an appealing and motivating experience to start learning programming.

Introduction

Failure rates in introductory programming courses worldwide motivated several authors to investigate the causes of such difficulties. Various reasons have been identified, such as the abstraction capabilities, the problem solving skills or the math's background required for learning programming (Gray et al., 1993, Jenkins, 2002, Teague et al., 2008, Winslow et al., 2005). These reasons bring forth many severe difficulties for the students in both understanding core programming concepts and in logical problem solving. These types of setbacks are particularly serious for an engineering student. A large number of individuals also report a lack of students' motivation, which provides a plausible explanation for the elevated percentage of dropouts.

We also agree that introductory programming courses require students to master highly abstract concepts such as data types, arrays and control structures. For those with this type of difficulty it becomes tricky to apply the concepts in the resolution of problems in multiple areas of knowledge. This fact, allied to the heterogeneous learning backgrounds and interests of each student, makes it almost impossible for the teaching staff to personalize the learning experience to meet each individual's needs. Nowadays, most of the students taking introductory programming courses still begin their learning experience by either constructing small pseudocode instructions or flowcharts. These methodologies provide a way to easily visualize algorithms. They also allow students to learn structures common to every programming language requiring little pre knowledge. However, the static medium in which they are used, usually paper, on the contrary, offers no direct feedback or dynamic visual cues to maintain student interest. This medium does not enable students to experiment and observe the effects of their own algorithms. On the other hand nowadays students are used to playing games with very appealing visual effects. So, we thought that the introduction of programmable worlds into the learning program might offer a great balance among motivation, exploration and practice, which would help students become better programmers. They provide the level of interaction necessary to keep students engaged and provide a familiar setting to many who already play video games daily. Through the use of programmable worlds knowledge can be presented at an adjustable pace and according to the difficulties of each individual, allowing room for improvement in the personalization of each learning experience.

Several pedagogical approaches and tools were proposed to help students learn to program. Many of those tools are based on animation and visualization of algorithms and programs. We can distinguish between specific tools, that only allow the animation of predefined examples and more general tools, that allow the animation and simulation of any solution (algorithm or program) developed by the student. In the first subgroup we can mention JHAVÉ (Naps, 2000) and in the second JELIOT (Ben-Ari et al., 2002) and SICAS (Gomes & Mendes, 2001). Others consist in simpler programming languages or mini-languages, designed to be simpler to use than traditional general-purpose languages. MiniJava is an example of this type of tool (Roberts, 2001). We

can also refer to the existence of controlled development environments, designed with educational purposes. In this type of environment the available options are limited in order to reduce the usual complexity of their professional counterparts. An example is BlueJ (Kolling, 2003). Micro-worlds are environments where the user has to ask some character to perform specific tasks in a simulated world, usually more concrete and close to the student context than other typical programming environments. Karel the Robot is a well-known example of this approach (Pattis, 1981). There are also tools to test student solutions to problems. Usually they try to compare the results of a student program with several provided input/output datasets. Among them we can mention WebToTeach (Arnow & Barshay, 1999) and ELP (Truong et al., 2003), systems developed for the Web. Although some positive effects were reported after the utilization of some tools, the number of students that drop out or fail programming courses is still remarkably high. From the knowledge we have of all these programs, we believe that the use of programmable worlds reveals very beneficial for learning introductory programming concepts. In particular, the ones like Scratch (Resnick et al., 2009) that allow not only the visual construction of the code, but also shows a strong relationship between the code constructed by the learner and its visual effects applied in the environment in question. It is very important for students to relate and explore the use of various concepts and, consequently, form ideas about its operation by observing the behaviour of various objects in the environment. Another idea that we consider very important is the fact that, as in Scratch, the environment should be developed with the objective of minimizing syntactic errors and should take advantage of the visual construction of control structures, small orders and assignments. We also consider essential for an early programming learning, the existence of simple and intuitive interfaces. These help the student to become familiar with programming concepts without the need to spend too much time understanding the functionality of the application. However, we noticed that there is a missing feature in the existing tools. Even environments such as Scratch, don't provide the student with a concrete goal beyond the construction of an interactive story or animation. This unlimited use can be confusing for inexperienced students. The lack of a fixed goal may cause the student not to use all the tools, working only with those that are familiar. Thus, some type of orientation is essential for the students to accomplish the task. We also believe that these tasks should be simple and familiar to the student. Thus, the idea of implementing the new environment named "Code Factory" is to provide a learning linear and interactive experience, where students begin by working with simpler concepts with well-defined objectives in order to acquire knowledge in a structured way and gradually progressing to more complex concepts.

Code Factory

The new developed environment, Code Factory, aims to be a friendly environment where novice students can explore core-programming concepts in a clear and motivating way. It consists of a platform style game where students can control a virtual robot not only to overcome multiple obstacles in a test of reflexes and logical thinking, but also have their first contact with the construction of small pseudocode instructions. There are several literature descriptions referring to the potential of games to motivate students (Garris et al., 2002, Malone, 1980, Papastergiou, 2009, Park, 2012).

The goal of this interactive game is to follow a friendly robot (Sparky) in his adventure to restore the order of his robot factory and its malfunctioning assembly lines and security systems. To do this, Sparky will have to count with the students help to navigate throughout the different factory levels. Each one of these levels has a malfunctioning system which must be fixed by correctly programming its behaviour and so that the student can complete the remainder of the level. The game alternates between environment exploration in a familiar platform gameplay and exercise solving where students can practice and improve their programming skills.

Code Factory provides an intuitive code building area where students can familiarize themselves with core programming concepts such as sequential, selection and repetitive structures, different types of conditions, operators and finally with the basic rules of code indentation.

The construction of pseudocode instructions is made in a highly visual manner; students can drag predefined structures into a grid comparable to the placement of pieces in a puzzle. Each piece contains a programming structure (Ex: If, While, ...) or an instruction relevant to the current exercise (Ex: Open, Close, Remove). Each time a new piece is placed, the game verifies if it's correctly placed and informs the student otherwise, allowing him to learn both structural and indentation rules early in the game experience.

By letting students freely experiment with the behaviour of the multiple objects present in the game environment, we hope to facilitate the development of a clear order-effect relationship with each created instruction and its consequences in the virtual world. Performance in each exercise is rated depending on the amount of tries used

to correctly solve it and bonus points are awarded accordingly. At this development phase, only some control structures are possible.

Pedagogic Considerations

The Code Factory also aims to provide an iterative and linear learning experience. The students begin by working with simpler concepts with well-defined objectives in order to acquire knowledge in a structured way, and gradually progressing to more complex concepts. The Code Factory was developed to provide an environment of simple and intuitive exploration. This allows students to have an active participation in the construction of their own knowledge making their first contact with programming positive and motivating. The use of exercises with a visual representation allied to the graphical building of pseudocode provides students with an interactive way to explore different programming concepts, enabling them to develop their ability to analyze and solve problems.

The key to provide a motivating but yet valid learning experience also lies in avoiding some of the harmful difficulties usually faced by students. By using predefined structures and instructions we eliminated the possibility of small syntax errors in hopes of limiting the students frustration when trying to conceive a valid solution. Equally important in the use of pseudocode is that it provides a simple way to construct algorithms through a more familiar language for the student however maintaining some proximity to a programming language. This pseudo language can be easily understood by individuals with no previous programming knowledge. Besides the elimination of syntactic errors this approach also helps the student to focus his concentration on understanding the problem and solving it. In order to stimulate the student, it is intended that this environment transit between activities of solving problems by building pseudocode and activities of playing in the 2D platform environment. The aim is to motivate the student to complete all the exercises/levels finding the best solution in order to collect as many points as possible in order to progress in the game.

To reach a greater number of students, independently of their learning backgrounds or personal interests, Code Factory uses exercises that rely greatly on visual cues, logical thinking and that require no previous knowledge from other courses. Sometimes the first programming exercises proposed to students have a high mathematical background without which students couldn't solve a problem. The scenario and objects used also make possible that the effects of the constructed code can also be observed in real time and context, without the need to interpret highly technical compiling messages.

It is intended that this environment work as a story with meaning, where the student will follow a friendly robot on its journey to restore order and harmony in their factory. To do this, it has to overcome the security systems and surpass faulty programming terminals of each area of the manufacture and assembly. At this point the student will have to solve the problem at hand in order to make each assembly line behave correctly, thereby restoring the functioning of the factory.

Code Factory was developed so that new information is conveyed progressively. When a new programming problem is presented, students have access to a brief explanation followed by similar examples of resolution. Exercises also increase in complexity, allowing students to familiarize themselves with a concept first in an easier setting. Each exercise has a clear objective, explained with a small text at the start of each malfunctioning system. With this feature we hope to provide students with enough information and independence so that they can explore each exercise on their own time and pace without the need of teacher support. The robot also has some features that enable it to improve their capacities and competences. As the students solve more problems, they earn more points. Students can then exchange their earned points with special items that improve the robot capabilities. In consequence, students that have better exercise solving performance can also overcome obstacles easier, inspiring students to strive for the best answer and to complete every given exercise.

Code Factory Interactive Environments

Code Factory is divided into two core interactive environments, the code construction environment and the game environment. These differ in presentation, purpose and user interface, allowing for the information and tools to be presented to the students only when necessary and creating a symbolic transition between the two different types of gameplay. In the code construction environment, students have access to the multiple tools needed to develop pseudocode instructions and fix each malfunctioning system while observing its effects on the virtual world. In the game environment students are provided with information relative to the robot state and can navigate throughout each level while avoiding the factories damaged security systems. Students can also collect points, which can be used to buy special items that improve the robot capabilities.

Code Construction Environment

The code construction environment (Figure1) is presented to students every time they are faced with a malfunctioning factory system (highlighted in Figure 1 with the legend "A Zone"). It is due to one or more mechanisms that prevent the progression of the robot. These mechanisms present in each level, behave erratically and it is up to the student to restore proper operation by building instructions in pseudocode. These are easily visually identified by an image of a computer terminal and also prevent the robot from venturing further into the current level. Students can only proceed to complete future exercises that feature more complex solutions or different programming structures after successfully restoring the correct behaviour of the current factory system. This behaviour is corrected by codifying the "fixing procedures" in pseudocode (B Zone of Figure 1). For a better understanding of the problem there is a small description of it and the student can also consult a small resolution example of a similar problem.

This B interactive zone provides every tool and information required to successfully solve each exercise. Here, students can conceive their solutions by manipulating each component of a pseudocode instruction and observe its effects in the A Zone and in the behaviour of objects presented inside it, each time it is executed. For a better interaction and so that it's easier and more intuitive for students to develop code, the interactive B Zone is organized in three smaller areas of interaction which organize tools by their function and similarity.

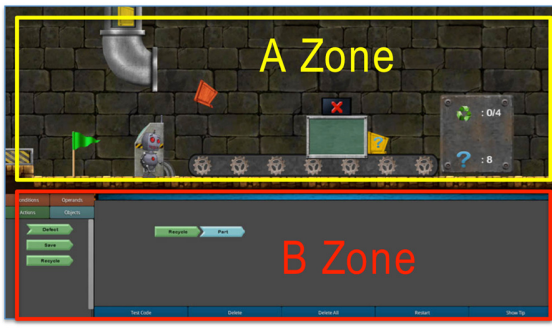

Figure 1: CodeFactory (Code Construction Environment) – Malfunctioning factory system (A Zone) and Code Construction Area (B Zone)

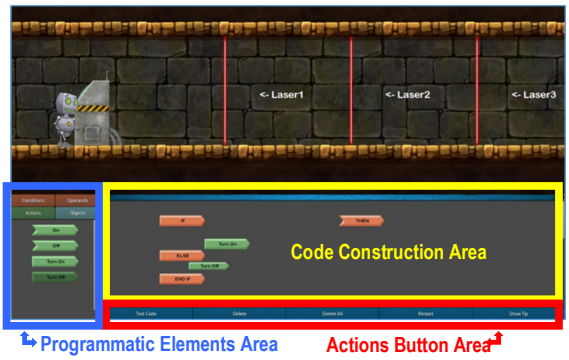

Figure 2: CodeFactory – Areas of B Zone

Following the textual and visual cues given by the environment, the student may then proceed to design a solution in the form of one or more pseudocode instructions. These instructions are then compiled and translated by applying a behaviour change mechanism. If the new behaviour fulfills the purpose of the exercise in question the student's response will be accepted and the student can advance. More points will be awarded to those who had fewer trials in solving the exercise. The B Zone of the Code Construction Environment, where all the tools are available consists of the following three sections highlighted in Figure 2:

1. ***Code Construction Area*** – This area functions as an empty canvas, where students can create and rearrange their answers by organizing programmatic elements. These can be dragged from a different interactive area, where the different elements are stored, until students are satisfied with the created solution. Then, by accessing the tools in the Action Buttons Area, students can verify if their answers have the desired effect, correcting the behaviour of the malfunctioning system and successfully completing the exercise.
   This area also provides constant feedback, helping and correcting students while they build an answer. Automatic code indentation inside programming structures helps students gain good code organizing skills and component placement rules prevent students from making basic mistakes.
   Although there is enough empty space in the canvas for students to be fairly liberal with their code organization, this area can be maximized to allow a greater number of programmatic elements in the screen at any given time. This will limit the space attributed to the visualization of the game environment and so it is better to return to the normal mode so that students can better observe the effects of their own instructions.

2. ***Programmatic Elements Area*** – From this area, students can access the different programmatic elements necessary to create a correct solution. It functions as a library and each element is organized in an easily identifiable separator. Students can drag elements to the Code Construction Area from any of the four existing separators (C Zone, Figure 2):
    a. ***Conditions.*** In this tab the student can find all the control structures necessary for the current exercise. From this tab it is possible to drag the elements (IF, ELSE or WHILE) to the construction.
    b. ***Operators.*** In this tab there are elements used to make comparisons between other elements, usually introduced as a result of a condition. The content of the separator is constant throughout all the exercises and consists of the following elements: EQUALS, DIFFERENT, AND, OR, GREATER THAN, LESS THAN, GREATER EQUAL, LESS or EQUAL.
    c. ***Actions.*** In this tab there are elements used to change the state of an object present in the game environment. These elements vary depending on the exercise and in this earlier stage are made available in accordance with the existent objects. The elements are: OPEN, CLOSE, RECYCLE, SAVE or REMOVE.
    d. ***Objects.*** In this tab there are the elements that identify the objects present in the interactive game environment (BOX, LASER, DOOR1, SENSOR2, ...). To these elements we can apply existing elements of the "Actions" so that they have certain behaviours. As in the tab previously mentioned only the elements that are part of the context of the exercise are available.

    In order to prevent syntax errors, the construction of pseudocode is done by dragging the different components (from the programmatic elements area to the pseudocode construction area) as if they were puzzle pieces. The application will only accept putting a new component if it respects the structural and syntactic rules of the pseudocode. If any of these rules are broken, the student will be immediately alerted of this fact, together with an explanation stating the reason for the inability to perform this operation. The elements will also be repositioned if the student puts them to any other position, even if they are logically correct, in order not to break the rules of code indentation. The student will also be alerted if any command previously constructed and executed by the application is found incomplete. The incomplete line will be marked with a red "x" on the left after the student tests the functioning of the whole code. However it is important to note that it is possible to have several different codifications to the same problem since they are syntactically correct.

3. ***Action Buttons Area*** – This area provides a set of simple tools to help students edit and test their answers, they can also clear the code construction grid, reset the exercise to its initial state and finally, display a brief explanation of the objective of the current exercise. It contains the following elements: “Test code”, “Delete”, “Delete All”, “Restart” and “Show tip”.
    a. ***Test code.*** After the construction of pseudocode and when the student wants to submit the answer, he/she should use this button in order to test if the solution is valid or not. The application will then perform the created pseudocode compilation. Then it will run automatically in case the pseudocode is correctly formulated. The student can then see a change in objects behaviour in the game environment according to the instructions built.
    b. ***Delete.*** This button allows the elimination of the last element put in the construction area. If any of the elements present in this area are selected, it will be eliminated regardless of the order in which it was inserted.
    c. ***Delete all.*** This button will delete all the elements in the construction area.
    d. ***Restart.*** In case the student, for some reason, wants to restart the exercise he/she can do so by clicking this button. Thus, all objects and mechanisms present and their behaviour will be reset to the initial state. However, the number of attempts the student has to find a solution will not be decreased but will be maintained.
    e. ***Show Tip.*** If the student has any questions about what is required in the current exercise, he/she can consult the exercise description and also an example of its resolution. The proposed resolution will not be a valid solution and only consists of a set of instructions to a similar problem.

Game Environment

The idea underlying the game environment is to engage student in the system, making him/her to be interested in the system, motivating him/her to progress in the game, overcoming obstacles, and consequently causing him/her the will to program. The game environment (Figure 3) presents and processes the interaction between the student and the robot. In order to progress further in the different game levels and to find each of the malfunctioning factory systems, students will have to observe the game environment, avoid the malicious objects that try to damage the robot and interact with the different platforms, doors and elevators that provide a safe path to the end of each level. In this environment the user can interact with the game world through the robot that will be able to move in two dimensions, trough the control of the directional cursor keys. At each level there will be various objects and obstacles arranged to attempt to prevent the progression of the robot. The student will have to move the robot to avoid these objects and complete the level successfully.

The user interface available in the Game Area provides information to the state of the robot health, warning the student if the robot is in danger of being fatally damaged. There is also an indicator, which shows the amount of points collected, and the number of special items currently owned by the robot.

Figure 3: CodeFactory – Game Environment

Scattered through each game level are small cogwheels that can be collected, these represent points that can be exchanged by special items that improve the robot capabilities. These are:

Health Item - When used this item restores twenty five percent of the total robot health. It can only be used if the robot is damaged.

Shield Item - This item partially shields the damages suffered by the robot by thirty percent. Each one of these items only works for the next three collisions.

Point Multiplier Item – When used this item multiplies the value of the next six coins collected by three.

Although the main objective of the platform game is to avoid that the robot suffers any damage in order to successfully restore the correct behaviour of his factory, accidents could happen. When the robot suffers fatal damage it will be returned to the starting point of the current level or to the closest activated saving point with a small decrease in the robots total health. Saving points, which are represented in the game world by a small green or red flag (activated and deactivated respectively), save the students' progress up to that point and can be activated by simply moving the robot to its location.

Development and Used Technologies

Code Factory's project requirements dictated that a high level language would be used for its development. The need to build a complex visual programming interface and a rich interactive game world, along with the requirement for multiplatform capabilities ultimately weighted in favor of the JAVA language. Its object-oriented nature allows a more comprehensible yet functional approach when developing the application's class structure. Its multiple external libraries also provide a much-needed help in implementing every feature.

This choice was also heavily influenced by the existence of the LibGDX framework, which provides an invaluable set of tools that facilitate multiplatform game development, along with user interface and graphical assets construction. This framework also encapsulates libraries such as the OpenGL ES 1.x/2.0, which allows a greater level of customization of the game's visual components.

Although the LibGDX framework was developed mostly in JAVA, its core methods and the resource complex features were developed in C++ for greater performance. Even though it possesses innumerous tools and libraries, the ones presented here played an extremely important role in the development of this application:

1. ***OpenGL ES 2.0*** – OpenGL ES is a subset of the OpenGL computer graphics rendering application programming interface especially developed for embedded systems such as smartphones and portable gaming consoles. It allows the construction and visualization of 3D polygons as well as the application of textures and shades to their surfaces. It provides developers total freedom to fully customize the game's visual style.

2. ***Asset Manager*** – This library greatly improves the memory footprint of the multiple application assets (images, audio files). It provides simple methods for the management of assets and also allows asynchronous loading so that the application remains responsive while loading the different assets.

3. ***Box2D*** – This library provides an easy and simple way to simulate physics and the behaviour of multiple bodies when influenced by external forces such as gravity. It also allows collision detection and processing. This library is extensively used in this application, mostly for robot movement and object interaction.

4. ***Scene2D*** – Similar to the widely used JAVA framework SWING, it provides the necessary tools and multiple components (such as buttons, text boxes and windows) for the construction of highly dynamic and interactive user interfaces. Interfaces build with this framework are completely multiplatform and can be easily adapted to different input types (touch, mouse).

Use Example

In the current state of development, Code Factory has twenty-three distinct levels, each of which contains a programming exercise belonging to one of the three main types (sequential, selection or repetitive structures). Each type is characterized by the use of a control structure and eventually a different mechanism whose behaviour differs from level to level. It is up to the student to observe and understand how these differences can be applied to reestablish the correct behaviour in each of them.

To better understand we will give an example where one operation of the fifth level is shown in Figure 4, including a simple selection structure. At this level the student needs to open a door so that the parts fall into the correct colour car. For this there is a sensor that will turn green for the door to open and drop the green parts. To solve this problem, the student can open the coding area, where he/she can read its explanation, which better clarifies the problem to solve. In this case, the description says the following: "The system that allows the correct type of part to be loaded is not working. The door needs to be open so that the green parts can be loaded into the kart."). The resolution will consist of the following. There is a sensor that is either red or green. When the sensor becomes green door1 should be open to let the green parts fall in the kart. The corresponding codification is shown in the Figure 5.

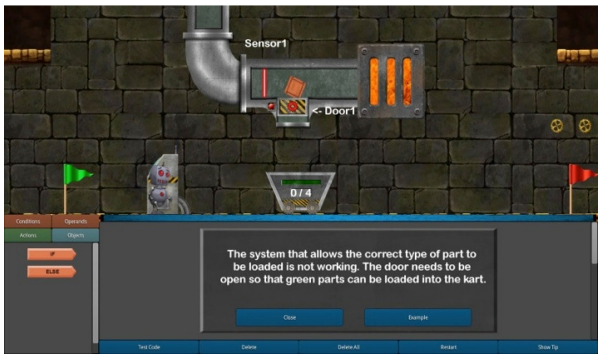

Figure 4: CodeFactory – Programming Activity description

```
IF Sensor1 Equals Green THEN
        Open Door1
ENDIF
```

Figure 5: CodeFactory – Programming Activity solution

Conclusions

We think that learning introductory programming can be facilitated by the existence of a tool to motivate students to programming activities. Current programming teaching includes a set of factors that makes the initiation of the students to the programming world a not so easy task. Apart from the particularly difficult nature of programming, its introductory teaching is not done in a suitable manner. Usually the teaching of algorithmic thinking and the necessary programming structures, even having a dynamic behaviour, is still done through static materials. Later, when students have to implement the solved problems, there is a huge load of syntactic details that students need to know. Additionally, the programming environments used are more suited for professionals, having no pedagogical features necessary for learning programming. We think that as educators we should manage ways to motivate students to learn fundamental aspects needed in any engineering programming course. Thus, this paper presents a new approach that we believe is very motivating for the placement of students in the programming world. At this moment this environment only includes a limited set of control structures and activities. However, in preparation there are a number of testing and future implementations.

References

Arnow, D., & Barshay, O. (1999). WebToTeach: An Interactive Focused Programming Exercise System. *29th ASEE/IEEE Frontiers in Education Conference*, San Juan, Puerto Rico. 39-44.

Ben-Ari, M., Myller, N., Sutinen, E., & Tarhio, J. (2002). Perspectives on Program Animation with Jeliot. *Lecture Notes in Computer Science*, 2269, 31-45. Springer-Verlag.

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

Gomes, A., & Mendes, A. J. (2001). SICAS: Interactive system for algorithm development and simulation. In M. Ortega & J. Bravo (Eds.), *Computers and Education in an Interconnected Society* (pp. 159-166). Kluwer Academic Publishers.

Gray, W. D., Goldberg, N. C., & Byrnes, S. A. (1993). Novices and programming: Merely a difficult subject (why?) or a means to mastering metacognitive skills? [Review of the book Studying the Novice Programmer]. *Journal of Educational Research on Computers*, 9 (1), 131-140.

Jenkins, T. (2002). On the difficulty of learning to program. *3rd Annual LTSN_ICS Conference*, Loughborough, UK. 53–58.

Kolling, M., Quig, B., Patterson, A., & Rosenberg, J. (2003). The BlueJ system and its pedagogy. *Journal of Computing Science Education, Special Issue of Learning and Teaching Object Technology*, 12 (4), 249-268.

Malone, T. W. (1980).What makes things fun to learn? A study of intrinsically motivating computer games. *Xerox Palo Alto Research Center Technical Report No. CIS-7 (SSL-80-11)*, Palo Alto, California.

Naps, T., Eagan, J., & Norton, L. (2000). JHAVÉ – An Environment to Actively Engage Students in Web-based Algorithm Visualizations. *31st SIGCSE Technical Symposium on Computer Science Education.*

Papastergiou, M. (2009). Digital Game-Based Learning in high school Computer Science Impact on educational effectiveness and student motivation. *Computers & Education*, 52 (1), 1-12.

Park, H., (2012). Relationship between Motivation and Student's Activity on Educational Game. *International Journal of Grid and Distributed Computing*, 5(1).

Pattis, R. (1981) *Karel the Robot: A Gentle Introduction to the Art of Programming*. John Wiley & Sons.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y. (2009). Scratch: programming for all. *Commun. ACM* 52, 11, 60-67.

Roberts, E. (2001). An overview of MiniJava. *ACM SIGCSE Bulletin Conference Proceedings*, 33 (1), 1-5.

Teague, D., & Roe, P. (2008). Collaborative learning: Towards a solution for novice programmers. *10th Conference on Australasian computing education (Vol. 78)*, Australian Computer Society, Inc., Wollongong, NSW, Australia.

Truong, N., Bancroft, P., & Roe, P. (2003). A web based environment for learning to program. *26th Australasian Computer Science Conference on Research and Practice in Information Technology (CRIPTS'03)*, Adelaide, Australia. 255-264.

Winslow, L., Bennedsen, J., & Caspersen, M. (2005). Abstraction ability as an indicator of success for learning object-oriented programming?. *SIGCSE Bulletin*, 38, 39–43.