Meet and greet programming using graphical languages and tangible interfaces

Pedro Dias
Computer Science Department
Institute Polytechnic of Tomar
Tomar, Portugal
pedrodias@ipt.pt

Sancho Oliveira
Instituto de Telecomunicações
DCTI - ISCTE – Lisbon University Institute
Lisboa, Portugal
sancho.oliveira@iscte.pt

Abstract — **Younger students are very often scared from programming, because of its complexity, before they even understand what it is or which purpose it serves. Using characteristics from different types of graphical languages and a tangible interface, Botbeans tries to present programming in a more casual and less complex way while keeping student's motivation high. This paper presents how different graphical languages were combined in order to build a simple and intuitive language with a tangible interface for students with low computer skills and that don't know what programming is, but have curiosity to try it.**

Keywords-component; programming learning, programming, graphical languages, tangible interface

I. INTRODUCTION

Learning how to program is complex, besides learning the syntax of the language, the apprentice needs to gain additional skills, like abstract thinking. Students that are naturally inclined to computers will make an effort and try to overcome this complexity to learn how to program, but students that don't have this predisposition will probably not try, scared by the complexity of programming.

There are multiple learning tools to help a student in the learning process of programming [1], the majority of these tools try to simplify the process of learning of how to program by removing complexity from the programming languages used, each tool with its own approach.

Botbeans was also designed to remove complexity from programming, but Botbeans in targeted to a different type of audience, a type of audience that doesn't know anything about programming and may not even know what its purpose is. This raises a new challenge, how to keep motivated an apprentice that does not yet know what programming is or if he is interested in it?

The first step towards a simplification was accomplished by removing all textual representation form the programming language. For this we have developed a hybrid graphical programming language; this language removes the syntax complexity of a textual programming language from the learning process, focusing the apprentice in learning how to build algorithms. In order to build a clean UI (User interface) while avoiding information overload, a Rich Client Platform was used in the development of the IDE (Integrated development environment), this allowed to build an IDE that displays information and tools as the apprentice request them.

Tangible interfaces have a good reputation in education [2]; they are capable of promoting motivation and are very often a catalyst for collaboration. In Botbeans a tangible interface was implemented in the form of a robot, this robot is the only source of inputs and outputs for the programing language. It is the interface of the apprentice's creations to the real world, letting them interact, using their senses, with their creations.

This paper will give details how the hybrid visual programming language was developed and implemented around the Rich Client Platform, and how the tangible interface work and affects the learning process while using the tool. Finally the results from an early test, done with a group of students during a science fair, are analyzed.

II. GRAPHICAL LANGUAGE

Graphical languages are very often associated with programming teaching [1], they allow removing the necessity of a new language syntax so student can understand concepts using their natural born language. The programming language is a mere tool that allows the programmer to write his algorithm in a format that a computer will understand, instead of teaching an apprentice a whole new language, graphical languages try to give them an intuitive language that they already know, enabling them to focus only on learning how to develop algorithmic thinking.

There are two main types of graphical programming languages, node-base and block-based. They are mainly different in their graphical representation, each one with different advantages and disadvantages.

A. *Node-based languages*

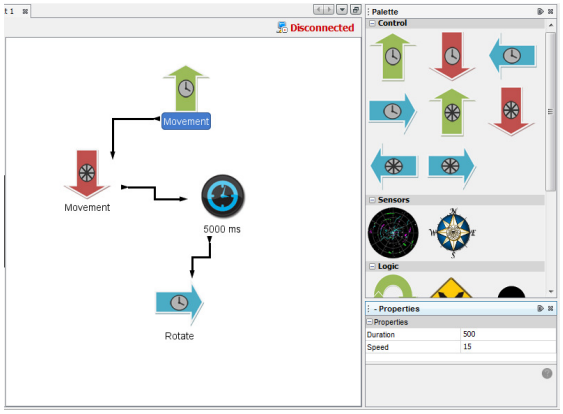

Figure 1. Node-based language in Botbeans.

This type of graphical language representation is very often associated with flowchart type designs; it consists in a group of different nodes (Table 1) connected to each other (Figure 1), in which each node has a different functionality and properties. The algorithm is executed by repeatedly evaluating each node and following the active connection for the next node.

In node-based languages the data and logic flow inside the implemented algorithm is very clear to the person reading it [3], allowing the student to easily debug its own implementation. However the representation of mathematical and logical expressions is very often [1] inserted inside specific types of nodes in the text format, which can be a problem if the student does not know how to read these expressions.

B. *Block-based languages*

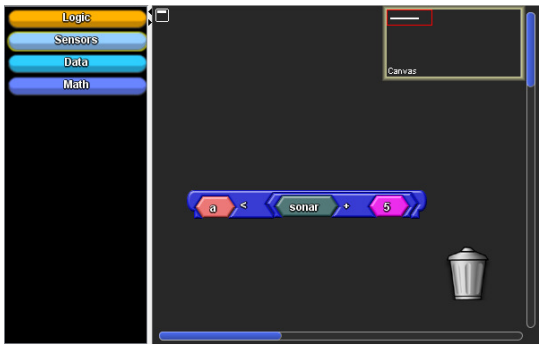

Figure 2. MIT's OpenBlocks based expression builder.

Block-based languages are represented by a construction made by grouping different blocks together (Figure 2). Similar to node-based each block has its own functionality and properties. In education this representation is very often associated with the creative process of children playing Legos.

This type of representation is very good for expressions and data types, because each block have a color and a shape and each shape corresponds to a different data type, allowing a natural differentiation of each data type by just looking at the blocks. Like a puzzle each block can only connect to blocks with compatible shapes, validating student's construction.

Data type identification and validation by block's color and shape is a good improvement from the textual representation in node-based, in terms of expressions.

C. *Hybrid solution implemented*

In order to remove all textual representation a hybrid solution was implemented using the best characteristics from node-based and block-based language types.

The language needed to expose the logic inside an algorithm in a very natural way, just like a river, where the program execution is the boat in the river. To accomplish that a node-based was developed, but instead of using a flowchart design, a graph design was used, this loosens the design giving more freedom and in the end inspiring the creativity of the student. If a loop is build, it will be clearly seen by the shape of the graph.

Each connection has one logic direction, which will be the direction of the execution; each connection made is verified, not allowing the student to do obvious logic errors, validating the algorithm at building time. This verification is done using two internal properties of each node that are, the maximum number of output and input connections each can have.

For example, a decision node, a typical "if", can only have one input connection and a maximum of two output connections, one which will be executed if the condition is true and another if it is false (Figure 3).

Nodes were categorized in five different categories:

Table 1. Nodes's categories

Control	Nodes to trigger robot's actuators making it move.
Sensors	Sensor input, each sensor have his own node.
Logic	Nodes that affect the logic flow in the algorithm. Ex: decision and union nodes.
Data	Variable definition and attribution.
Misc	Timer, speaker and other miscellaneous nodes.

Since this node-based language had text mathematical and logic expressions inside some specific nodes and block-based language had a good definition of expressions a hybrid solution was implemented. An expression builder was developed using the block-based language, outside this builder each expression is still displayed inside a node but now using blocks instead of text.

The expression builder was not developed from scratch; instead it was based on MIT's OpenBlocks Framework [4]. This framework recently got some recognition by Google, which used it for their Google App Inventor, a tool that lets users with low programming skills build Android applications.

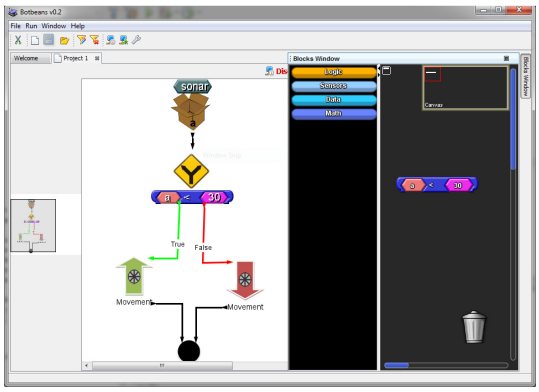

Figure 3. Botbeans and Openblocks integration.

To avoid information overload in the user interface, the expression builder is always hidden and it is only shown if a student need to edit an expression by double-clicking it. This tries to avoid the student to be distracted by multitasking between both graphical representations.

III. TANGIBLE INTERFACE

Tangibles interfaces allow their users to involve in a more deep way in the task they are trying to realize [2], this is very important in learning how to program. If the student can interact physically with his creation, motivation will rise and more important he will start losing the fear of experimenting new ideas for his implementations, boosting creativity.

Typical programming languages require the programmer to be able to understand the results the algorithm just generated; this is not always an easy task for a beginner and in the end can make him insecure. If he can experience the results by felling them with his senses (see, touch and ear) he will more easily understand the results [5].

With this tangible interface through the robot interaction with the real world, the result's interpretation are transformed in an observation of the robot's behavior, if the robot starts misbehaving, instantly the student will know something is wrong and that he's algorithm is failing. Another good characteristic of this type of interaction is that since the tool have two points of interaction, the computer screen where the algorithm is implemented and the robot where the inputs and outputs of the language are, collaboration starts happening more naturally because it's possible for multiple students interact with the tool simultaneously.

The inputs of the interface are in reality the robot's sensors, like distance, sound and light sensor, when input nodes are executed in the graphical language the sensor data is requested from the robot and used internally by the language. Outputs are the motors for movement, internal speaker for sound and finally robot's display.

IV. EXPERIENCE

In 2010 during the "Festa da Ciência, Cultura e Tecnologia", an annual science fair at Institute Polytechnic of Tomar, an experience was built (Figure 4.) where some of the students visiting the science fair were invited in trying to play with Botbeans. An immediate collaborative behavior was evident between the visitors. Very often when a student started executing the algorithm that he had just implemented, another student entered the scene giving suggestions to his colleague on how to improve the algorithm or try something different. The robot was the big motivator since during the execution one of the students was positioning the robot and the other one was at the computer. If the experience was in some place familiar to students, like their own school with their own teacher, results should probably be even better.

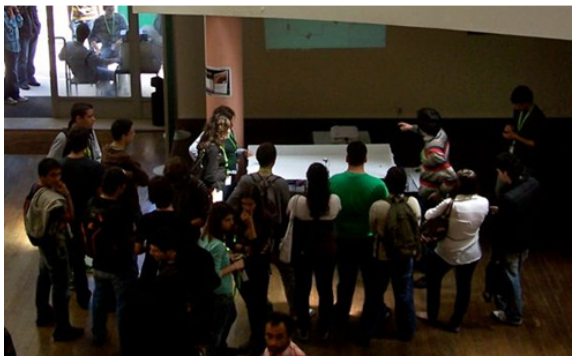

Figure 4. Experience at FCCT 2010.

The white table visible in Figure 4, was the area available for the robot to move, inside were put some obstacles, which students could move to build their own testing facility. The graphical language was projected to the wall in front of the table, projecting the graphical language was very positive because students could watch the execution of each node in their implementation and simultaneously, see how that affects the robot in the real world.

V. IMPLEMENTED SOLUTION

A. *Robotic Platform*

Current version of Botbeans only supports one robotic platform, which is Lego Mindstorms NXT. The decision to support this platform before any other was related with the fact that this platform is already present in many education facilities.

The modular architecture in the tool allows in the near future supporting more robotic platforms and to extend the functionalities of the already supported platforms, like adding support for more sensors or motors. The only obligatory requirement for the robotic platforms is being able of some type of wireless communications, this is needed because the computer needs to be connected to the robot in order to show the student which nodes are in execution and sensor data.

Since Botbeans is able to implement support for new robotic platforms, more advanced students can even build small simple robots and use them with the tool.

B. *Rich Client Platform*

Students are a very special type of users, they will do things that the tool developer never thought about, sometimes without even knowing what they are doing and this requires a learning

tool to be really bug proof. With this in mind Botbeans needed to rely on a good tested base, the usage of a Rich Client Platform gave Botbeans some of the basic IDE's functionalities, like drag-n-drop, which were already tested by the RCP developer.

RCP development also allowed having a rich interface which was important to accomplish the requirement of maintaining the GUI clean and without information overload to its user keeping him focused in programming but simultaneously giving him all the tools he needs but only when he needs them.

VI. Other Tools

There are many other learning tools for programming, but Botbeans is aiming for a different target audience. A lot of these tools are aimed for students that are learning how to program but already have an idea of what is programming and what its purpose is. Botbeans is aimed at students with an inferior age and that don't know what programming is and more important they don't know yet in which area they will focus their studies in the future. This is why keeping the student motivated is so important for Botbeans, it is a lot harder to maintain a student that's trying programming for curiously and not because he has to or it is one objective he has to accomplish.

Each tool has its own approach; there are tools like BlueJ and Greenfoot used to teach more advanced topics in programming, like object oriented programming [6]. Some of these tools require their users to have already some knowledge of the basic programming structures, since their purpose is not to teach these structures to people without any knowledge or interest in programming. Other tools like Portugol [7] try to help the student to take the jump to an industrial programming language by keeping a graphical language and a textual language in his natural language synchronized. Botbeans instead tries, in a casual way, to show to a student that only has basic computer knowledge what programming is and how interesting it can be, helping him taking the jump to the next level which can be one of the previous tools.

VII. Conclusion

Botbeans was designed to show students of small ages what programming is, without the complexity that usually is associated to it, and which usually scares students away before they even know what is programming. This tries to reduce the steep in the learning curve using a combination of teaching techniques, techniques to help reducing the complexity associated with programming, and techniques to keep the student motivated, simultaneously sparking collaboration.

The development of the hybrid graphical language enabled Botbeans to simplify the programming language leaving only the essentials programming structures, since the language is graphical these structures are presented to the student in a format he is naturally already familiar with.

Without a motivated student the tool would have limited success, the interaction between the student and the robot, motivates the student and sparks his creativity to try new things. Robot can be used as output using its motors, speaker and display or input using its sensors, this enables a bidirectional interaction between the student and the tool.

This tool is still a work in progress but its modular architecture will allow in the future to expand the number of functionalities supported without increasing the complexity of each one.

References

[1] C. Kelleher and R. Pausch, "Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers," ACM Computing Surveys (CSUR), vol. 37, Jun. 2005, pp. 83–137.

[2] B. Schneider, P. Jermann, G. Zufferey, and P. Dillenbourg, "Benefits of a Tangible Interface for Collaborative Learning and Interaction," Learning Technologies, IEEE Transactions on, vol. PP, 2010, p. 1.

[3] B. Smith, "Conceptual graphs as a visual programming language for teaching programming," Visual Languages and Human-Centric Computing, 2009. VL/HCC 2009. IEEE Symposium on, 2009, pp. 258-259.

[4] R.V. Roque, "OpenBlocks : an extendable framework for graphical block programming systems."

[5] C. Panadero, J. Román, and C. Kloos, "Impact of learning experiences using LEGO Mindstorms® in engineering courses," Education Engineering (EDUCON), 2010 IEEE, 2010, pp. 503-512.

[6] Xinogalos, S.; Satratzemi, M.; Dagdilelis, V.; , "Re-designing an OOP course based on BlueJ," *Advanced Learning Technologies, 2007. ICALT 2007. Seventh IEEE International Conference on* , vol., no., pp.660-664.

[7] A. Manso, C. Marques, and P. Dias, "Portugol IDE v3.x: A new environment to teach and learn computer programming," Education Engineering (EDUCON), 2010 IEEE, 2010, pp. 1007-1010.