An Interactive Book Authoring Tool to Introduce Programming Logic in Schools

André Campos[1], Alberto Signoretti[2] and Mário Rodrigues[3]
[1]*DIMAp, Federal University of RN - UFRN, Natal, Brazil*
[2]*DI, State University of RN - UERN, Natal, Brazil*
[3]*ESTGA/IEETA, University of Aveiro - UA, Aveiro, Portugal*

Keywords: Computational Thinking, Storytelling, Digital Interactive Book, Computer Science Education.

Abstract: In the past years, there was a growing interest in teaching computational thinking in elementary and high school institutions. Although the idea is spread and well accepted among academics, it has been rarely put in practice in the classrooms. Currently, when a programming-related activity is offered, with some few exceptions, it is usually presented as an extra-curricular (optional) activity. However, it does not need to be disassociated from the common school curriculum. The present work is based on the idea that programming logic can be used transversally with different subjects, such as history, geography, science, literacy, mathematics, among others. The authors envisage to accomplish this goal by enabling programming as a supporting tool for teachers and students, allowing them to create digital interactive books. The tool, named piBook, has its main focus in the production of interactive storytelling using non-linear narratives. Besides, it is also possible to create textual games (such as role-playing games), interactive activities (such as quizzes), tutorials, chatbots and similar applications.

1 INTRODUCTION

Recent articles has shown the importance of aligning the education of "digital natives" (Prensky, 2009) with the development of digital skills and competences (Leonard et al., 2016). However, a considerable number of schools still do not address this need. In several schools, computers are being primarily used as tools for information research, text editing and presentations. In some contexts, computers are also used as a motivational tool for learning, through the use of games and playful digital activities (Felicia, 2014). However, even in these latter cases, there is no direct stimulus for the students to know how these games and applications are built.

Knowing how to use tools is the most basic relationship one can have with technology, which differs significantly from knowing how to apply concepts to construct them. In the today digital society, this ability is highly appreciated in any career. Furthermore, knowing or identifying the logic behind a computer-based solution can help the student's autonomy to solve objective problems by developing the ability to think in a systematical and critical way (Wing, 2006).

Ten years after Jeanette Wing's influential article, where the term Computational Thinking was coined (Wing, 2006), the idea has still not been widely adopted. Despite the growing interest in the subject, there is a huge difficulty in putting her ideas into practice, especially in countries with strong resistance to change or to provide more flexibility to their scholar curriculum. As a consequence, several educational institutions that adopt computational thinking and correlated activities, usually do so through extracurricular activities, following the learn-to-code movement (Kafai and Burke, 2014).

Several institutions and entities worldwide have been supporting this movement in the last years. This is mainly due to the stimulus that programming offers to logical thinking, creativity, reasoning and problem solving through abstractions and decompositions (Lye and Koh, 2014). Therefore, it is critical to spend efforts in introducing computational thinking and, more specifically, programming logic in the early years of education.

The present work is complementary to the existing learn-to-code initiatives in extracurricular activities. However, differently from them, it considers that teaching programming logic does not necessarily have to be disconnected from the current existing

Campos, A., Signoretti, A. and Rodrigues, M.
An Interactive Book Authoring Tool to Introduce Programming Logic in Schools.
DOI: 10.5220/0006333501400148
In *Proceedings of the 9th International Conference on Computer Supported Education (CSEDU 2017) - Volume 1*, pages 140-148
ISBN: 978-989-758-239-4

school contents. In other words, it can be transversally used on diverse subjects, such as history, geography, science, literacy, mathematics, foreign language, among others. For this, it is necessary to embed the use of programming logic into the existing learning activities, so that it can be part of a teaching method.

A widely accepted "general purpose" teaching method is the use of storytelling. According to Robin (Robin, 2006), storytelling and more specifically interactive digital storytelling is a powerful teaching and learning tool for engaging both teachers and students. Besides, storytelling can effectively be used in many different contexts and subjects. A language teacher can, for instance, ask his/her students to create a non-linear narrative in order to improve their literacy skills (Menezes, 2012), while a mathematics teacher may aim at developing the student's argumentative competences (Albano et al., 2016). These examples show how flexible is the use of storytelling in the educational context.

The present work exploits the flexibility of the storytelling to promote computational thinking and programming skills in the existing scholar curriculum. This is done with the support of a new tool, named piBook (Programmable Interactive Book), which the current paper means to introduce.

The paper is structured in five sections. The current one exposes the motivation and goals of the work. The second section presents correlated strategies used to create digital non-linear narratives. The following one provides the piBook goals and main requirements. The next one details the piBook system, by presenting its architectural elements and how programs are created on it. Finally, the last section provides some considerations about the developed tool as well as intended future work.

2 RELATED WORK

Some computational tools have already been created with the purpose of enabling users to create digital interactive stories as well as to publish their books. Most of these tools use narrative flows (i.e., directed graphs) in order to define the non-linear structure of the story. However, some non trivial interactions are particularly difficult to accomplish without the use of programming resources and concepts (such as variables, conditionals, loops). While some tools deal with this problem through a purely visual approach, others allow authors to create a more complex logic by directly using programming languages.

An example of the first case is the StoryTec (Gobel et al., 2008). StoryTec is storytelling platform composed of two major components: an authoring environment and a runtime engine. The authoring component is based on a pluggable framework where different editors may be used to create elements for a story. It was initially conceived with five components: a Story Editor, a Stage Editor, an Action Set Editor, a Property Editor, and an Asset Manager. The editors directly related to the current work are the Story and Action Set editors. While the first one provides an interactive 2D representation of the story graph, containing the scenes, the transitions between the scenes and the story elements, the second one is used to configure the story logic inside each scene, which is made through a visual programming environment. The scene logic is then described visually by a set of rules composed of conditions and actions. The programming approach of this editor is similar to a flow diagram, as illustrated in Figure 1.

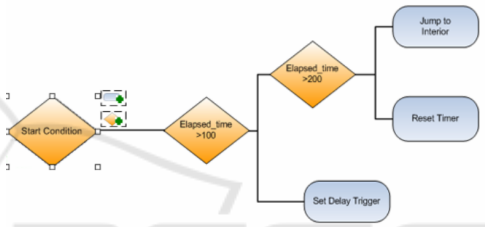

Figure 1: Example of programming rules in the StoryTec's Action Set Editor.

The StoryTec's visual approach eases the use of the authoring tool for non-specialists due to the fact that no programming skills are required to define the story logic. Nevertheless, it restraints the ability to create rules for complex scenarios. A more recent storytelling authoring tool, named Twine (Hahn, 2016), tackles this gap by providing the ability to write the story logic embedded in the narrative by using a textual programming language. The way in which Twine creates non-linear structures in the story is based on wiki systems. The user creates a text, which may have one or more wiki link to other texts, which may also connect to other ones, and so forth. For increasing usability, Twine provides a visualization tool where the user can see the overall story graph, in a way similar to StoryTec. However, differently from the latter, it does not provide a visual editor for authoring the story logic. In Twine, each wiki-text may also have embedded a chunk of code (written in Javascript) in order to control, for instance, how many times the user has passed by that text or if a part of the text should be visible or not. This approach allows more control. However, it requires previous knowledge of a specific programming language. Figure 2 illustrates the Twine approach to embed variables and coding instructions into a narrative text.

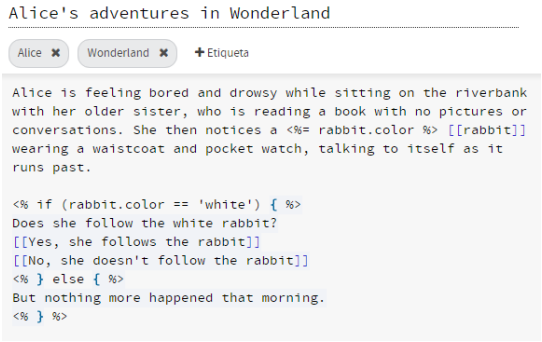

Figure 2: Example of logic embedded in the narrative in Twine.

The two previously mentioned approaches, exemplified by StoryTec and Twine, have advantages and pitfalls. While a purely visual method does not provide all programming mechanisms, a direct use of a programming language may pose a barrier to beginners. It is worth then note that each approach targets a particular audience: non-programmer users and programmer ones. By focusing on a unique audience, they do not provide enough mechanisms to guide a non programmer user to become a programmer one. In other words, they are not adequate to be used as a learning tool.

Aiming at this learning guidance in programming skills, several tools have been constructed using a block-based visual programming strategy, as the one present in Scratch (Resnick et al., 2009). Indeed, the interest in block-based visual programming has grown in recent years, specially in the educational context, helping beginners to get involved in their programming first steps (Fraser, 2015). From a leaner perspective, the advantages of using block-based programming relies on the ability to use a visual language modeled for a specific domain, the possibility of gradually presenting parts of that language, and, finally, the capacity of not allowing syntax errors (only semantic errors) (Fraser, 2015).

Because of the previously mentioned reasons, authoring applications using block-based programming, such as Scratch (Resnick et al., 2009) and AppInventor (Wolber, 2011), have been widely used by educators to teach programming. However, concerning to the current work, although it is possible to create non-linear interactive stories with Scratch, AppInventor and similar applications, their visual languages do not target the specific purpose of creating non-linear interactive narratives. In fact, they can be categorized as general-purpose authoring tools. A tool targeting this particular application context may facilitate the authoring process, while helping teaching programming logic.

The present work correlates the three previously mentioned approaches in a unique learning platform. It provides a tool where the user can visually author a non-linear sequence through a graph-based flow, but he can also program the story logic by using a block-based strategy or, when he becomes more proficient, a textual programming language.

3 GOALS AND REQUIREMENTS

The primary pedagogical goal of the current work is to stimulate the development of computational problem-solving skills in the educational context by the challenge of authoring digital interactive storytelling. This goal has however a natural barrier: school teachers usually do not have previous background on computational thinking nor programming skills to set up a logic behind their non-linear narratives. Taking into account that barrier, it is imperative to target different types of audience, from novice to expert users. So, the tool should be easy to use by newcomers in programming (programming skills should not be required), but also to be expressive enough to programming experts. Besides that, as it is intended to be used in an educational context, it should also provide a smooth transition from the novice to the advanced one.

From the novice perspective, users should be able to conceive their non-linear narratives visually by creating states in a graph-based flow-oriented diagram. Each graph node is considered as state in the story, which can only go forward or backward according to the reader's choice. For the intermediary users, they should be able to express more complex mechanisms by visually dragging and dropping programming blocks (Fraser, 2015). The latter introduces programming concepts useful for handling more complex reader interactions. Finally, as the intermediary users become more proficient in the concepts of programming blocks, they may turn into advanced users and change their programming environment to a textual programming language. The latter should provide the possibility to fully customize the reader interactive experience.

Besides the aforementioned goal, the following requirements were also established in order to set up a playful educational environment around the tool:

- Ability to collaboratively create the story contents, including the programs controlling the logic behind the story interactions;
- Ability to share the created story with other users (e.g. classmates and parents) and get feedback from them;

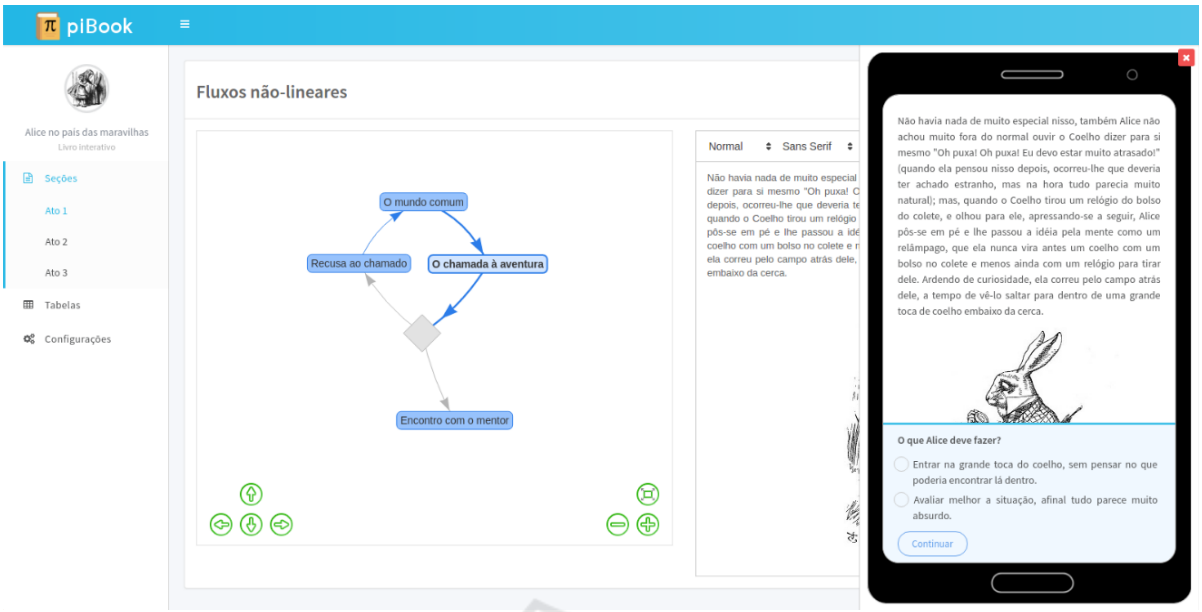

Figure 3: piBook screenshot showing a flow-based programming mode and a preview of the interactive book.

- Ability to publish a final version of the story (interactive digital book).

These items resume the scope of piBook. Nevertheless, it is worth to highlight that piBook has not been designed to be a self-guided learning tool. A self-guided or a self-discovery learning tool must follow a learning method and, for the moment, there is no particular methodological approach set up to piBook. Therefore, researchers and educators may explore the pedagogical possibilities (online and offline) and students are supposed to be oriented by them to use piBook as a way to put their skills and knowledge in practice.

4 piBook

piBook is an online application composed of two parts: an authoring platform and a running application. The first one is a web-based system where users, students or teachers, may create interactive books for their audience, the readers. The second part is a mobile-based application able to load an interactive book description and execute it. Figure 3 shows a screenshot of the authoring platform in flow-based programming mode. The figure also presents a preview panel where the user can check his creation before publishing.

For convenience, we refer hereafter piBook authoring plataform as *piBook tool* and the books authored by the application just as *piBooks*.

4.1 Elements

In order to understand how to author a piBook, it is important to have an overview of the elements composing the tool and the authored books. A piBook is structured into four main elements, which are:

- Section: sections provide a way to organize the book, either to give sequence of a narrative, like chapters, or to give a feedback of evolution in the activities, like game levels. A user/reader may, for instance, stay in a section (level) while some goal has not been accomplished. As soon as the reader reaches the section (level) goals, another one can be shown.
- Formatted texts: these refer to the visual contents of the application, which are formatted texts optionally embedded with images, videos, math formulas, and references to data stored in spreadsheets or program variables. Formatted texts are then visual templates, which are used to interact with readers.
- Spreadsheets: piBooks may use persistent data to fulfill the book's templates (formatted texts) or to save data from reader interactions. Spreadsheets are where these data are saved. The reason for using spreadsheets instead of database models is to ease the edition and generation of reports. Since book authors may not be programmer experts, they may benefit from using a common and well-known interface element as a spreadsheets.
- Programs: they define the logic of the interactive

book, choosing the template to show to readers, fulfilling it with corresponding data, and deciding what to save in the spreadsheet after a reader interaction. It is the kernel of a piBook.

Theses elements are illustrated in Figure 4. The numbers in the figure represent the cardinality of the relations between elements. A piBook is composed of one or more sections, formatted texts and optional spreadsheets. Each section should have an associated program, from which the formatted texts and spreadsheets are handled.

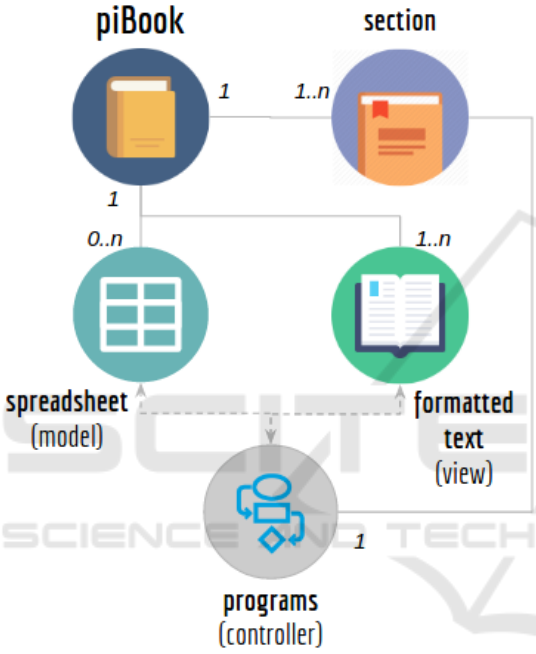

Figure 4: piBook elements.

The figure also highlights the correlation between the spreadsheets, formatted texts and programs with the architectural pattern known as MVC (Model-View-Controller). This pattern creates a separation of concerns by assigning responsibilities to different elements. While the formatted texts form the communication channel with the reader (the view), the spreadsheets store useful book data (the model), and the program controls the logic behind the scenes (the controller).

Although the usage of the tool was primarily targeted to create narrative stories for an educational audience, it was perceived during its development that several other applications could be also constructed through the use of its concepts and resources. Possible uses involve the creation of different kinds of interactive activities, such as games, quizzes, tutorials and chatbots. Figure 5 illustrates some of these uses.

Figure 5: Examples of pi-Book applications targeted on mobile devices.

4.2 Programming Modes

As the primary idea behind the tool is to promote computational thinking and programming, a piBook author is not supposed to have previous knowledge about computer programming. It is then necessary to build a smooth learning curve. Our approach to introduce computer programming relies on three levels of ability.

The first one was conceived to the users who have never programmed computers using a formal computer language. In this level, a piBook logic is defined visually by a flow-oriented state-based diagram. The logic is then composed of several states, each one related to a specific formatted text. When the book is in a state, the corresponding formatted text is presented to the reader. After the presentation, the book may go to a new state, including a user interaction state, like an option request. When the user chooses, the book

switches to a new state, and so on. Figure 6 illustrates an example of flow-based program. Starting in state T1, the formatted text associated to T1 is presented to the reader. After that, the book goes to state T2 and its corresponding text is also presented. Then, a reader interaction is requested, asking to choose an option through state O3. According to the answer, the book goes to state T4 or T5. After T4, a new user interaction is requested and the book state can switches back to state T1 or T2.

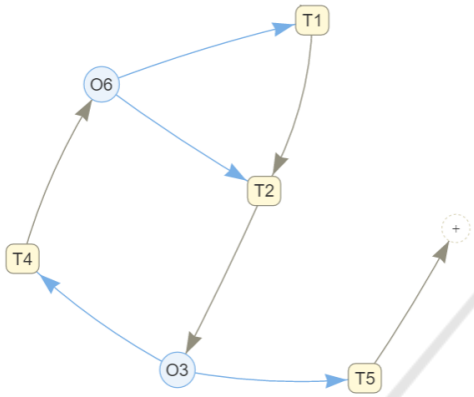

Figure 6: Flow-based program.

As previously mentioned, this level of logic programming is well-suited for beginners, since the visual approach facilitates the conception of non-linear stories even for non programmers. Besides that, the book author can have a glimpse on the overall book and when its content flow changes. However, this approach is also limiting. It restrains the use of more labored interactions. Consider, for instance, the book's author wants to get the reader's name in order to provided personalized feedback during the reading. It will not be possible with the flow-based approach. For this, it is necessary to deal with the concept of variable, which will temporarily store the reader's name.

The second programming level tackles this limitation. It was conceived to the users who want more expressiveness in their books, but have not enough experience with computer programs yet. In this level, a piBook logic is also defined visually, but with a block-based domain specific language (DSL).

As one of the designed goals of piBook tool is to provide a smooth learning path between user programming expertise, a program written in a flow-based way should be able to be translated to the block-based mode. Therefore, the state transitions of flow-based programs should also be present in a block-based solution.

Flow-based programs are Finite State Machines (FSM). As so, it naturally handles state transitions. To provide the corresponding mechanism of state transitions in block-based programs, an event-based strategy was chosen. This approach takes into account the events triggered by actions performed by readers in a particular state. As soon as an event is triggered, a new state transition occurs and piBook enters another narrative state.

Following this approach, if the user who created the flow-based program shown in Figure 6 wants to start working with block-based program, he can translate his original flow-based code to a block-based one. Figure 7 illustrates an extract of the block-based code resulting from this transition. In the figure, T1, T2 and Q1 represent the corresponding states and the reader interaction state 1 in Figure 6.

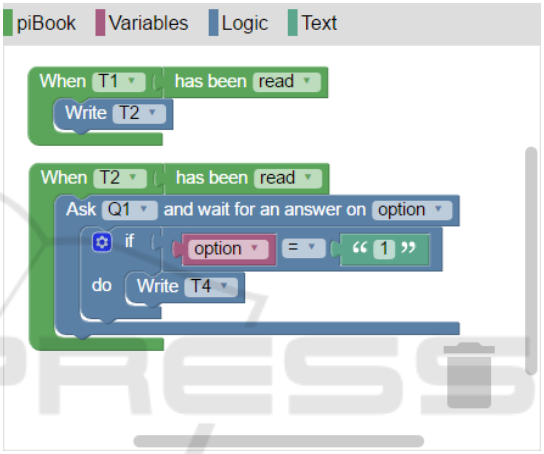

Figure 7: Extract of a block-based program.

After that transition, the user can improve his block-based code by using programming resources such as variables, conditionals and loops. At this point, the user can, for instance, iterate over a previously saved spreadsheet data in order to check a reader answer.

Finally, the third level enables the book authors to conceive more complex interactions with the reader. In this level, piBook logic is defined as a textual general purpose programming language. The current supported language is Javascript, but other scripting languages may be incorporated depending on the needs of the future piBook users.

To facilitate the translations between programming modes, the textual mode also follows the event-based strategy incorporated in the block-based mode. Again, if the user who created the flow-based program shown in Figure 6 wants to start working with textual-based program, he can translate his original code to a general purpose programming language as the one presented in Figure 8. The states T1, T2, T3, T4, Q1 and Q2 represent the same states and the reader interaction states as described in the block-based mode.

```
01.  book.write(T1);
02.
03.  T1.on('read', ()=> book.write(T2));
04.  T2.on('read', ()=> book.ask(O3));
05.  T4.on('read', ()=> book.ask(O6));
06.  T5.on('read', ()=> book.end());
07.
08.  O3.on('answered', (answer) => {
09.      if (answer == 1)
10.          book.write(T4);
11.      else
12.          book.write(T5);
13.  });
14.
15.  O6.on('answered', (answer) => {
16.      if (answer == 1)
17.          book.write(T1);
18.      else
19.          book.write(T2);
20.  });
```

Figure 8: Code example based on event transitions (in Javascript).

These three programming levels are assumed to handle the different levels of knowledge of the targeted audience, from a beginner student to an experienced one. However, it is worth to note that, due to expressiveness capacity of each mode (or its limitations), the tool allows a code to be translated from a less expressive mode to a more expressive one, but not otherwise. In other words, if the user starts a section using a flow-based program, he can transform it into block or textual-based code. However, after that, he cannot revert a block or text-based code back to a flow-based program. The same is valid between block and textual code.

4.3 System Architecture and Data Format

As previously mentioned, from the user perspective, piBook is an online application composed of two parts: an web-based authoring tool (book creator) and a mobile application (book reader). There is however a third component, a web server, to handle persistent data and to provide visible online data (published books). The server exposes a JSON-based REST API used by both the authoring tool and the mobile application to create, read, update or destroy book properties, sections, contents or logic specifications.

The authoring tool is a Single Page Application (SPA) composed of three main components: 1) a user dashboard, where the user can edit his authored books, create new ones and publish them; 2) a book editor, where the user can edit the book sections, their contents and logic; and 3) a data editor, represented by a spreadsheet where the user can collect data from book interactions and/or serve static data for his book.

The authoring tool, as well as the mobile application, consume data from the server in a specific format. The data format exchanged by server and applications induce the hierarchical structures of a piBook into sections, and the latter into a collection of text snippets and a section logic. The snippet contents follow the Delta format to describe Rich Text Documents (RTD) specified by the Quill editor (Quill, 2017).

Figures 9 and 10 illustrate how a book specified in the flow-based mode is represented by the specified data format. Such format represents the book's contents in a static way. The data is then interpreted by the mobile reader to dynamically construct the interactive application.

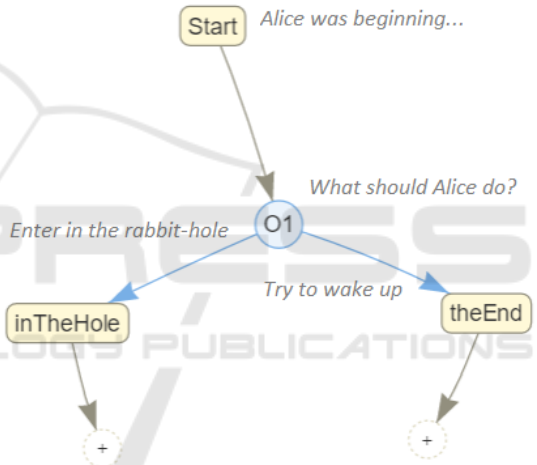

Figure 9: Example of flow.

5 FINAL REMARKS AND FUTURE WORK

The current paper introduced a tool, named piBook, conceived to help the introduction of programming logic in schools. The paper describes the ideas of the tool, how it is structured and the reasons behind the choices. The choice of focusing on storytelling was mainly due to the flexibility it provides, enabling the introduction of programming logic in most of the subjects of a scholar curriculum, from history to sciences.

The tool is not production-ready. For the time being, it is possible to specify interactive narratives in the flow-based programming mode, by writing formatted texts, options requests, and defining the logic to control these contents and interactions. However,

```
{
  "title": "Alice's Adventures in Wonderland",
  "authors": [ "lewiscarroll" ],
  "sections": [{
    "title": "Down the Rabbit-Hole",
    "snippets": [{
      "type": "text",
      "label": "Start",
      "content": [
        { "insert": "Alice was beginning...\n" },
        { "insert": { "image": "rabbit.png" } }
      ]
    },
    { "type": "options",
      "label": "O1",
      "content": [
        { "insert": "What should Alice do?\n",
          "attributes": { "header": 2 } },
        { "insert": "Enter in the rabbit-hole\n",
          "attributes": { "list": "bullet" } },
        { "insert": "Try to wake up",
          "attributes": { "list": "bullet" } }
      ],
      "options": {
        "o1": "Enter in the rabbit-hole",
        "o2": "Try to wake up"
      }
    },
    { "type": "text",
      "label": "inTheHole",
      "content": [
        { "insert": "Alice entered...\n" }
      ]
    },
    { "type": "text",
      "label": "theEnd",
      "content": [
        { "insert": "and Alice woke up.\n" },
        { "insert": "The End",
          "attributes": { "header": 1 } }
      ]
    }],
    "logic": {
      "flow": {
        "transitions": [
          { "from": "start", "to": "O1" },
          { "from": "O1", "option": "o1",
            "to": "inTheHole" },
          { "from": "O1", "option": "o2",
            "to": "theEnd" }
        ]
      }
    }
  }]
}
```

Figure 10: Example of JSON data representing the contents of the book specified in Figure 9.

transitions to block and textual modes are still in development. Usability tests in the flow-based mode is currently being made in order to improve user experience.

The presented tool is not been conceived to be a self-guided learning tool. It just helps students to put in practice previously acquired knowledge in order to develop their coding skills. It is supposed to be used as a supporting mechanism for different teaching strategies. However, as a companion of storytelling, piBook seems to be well suited for project-based learning initiatives (PBL) (Krajcik and Blumenfeld, 2006), where students are designated to author interactive contents for a specific subject and, for this, they must produce the logic behind the interactive activity. However, an adequate methodological approach for piBook has not been conceived yet. Usability tests with school teachers should still be done prior to applying experiments with students.

Besides the improvements and requests after the usability tests, some pedagogical evaluations must also be performed. An assumption is that the existence of different levels of programming modes facilitates project-based learning activities, since the same project can be assigned to the whole class and each student may chose a level according to his/her prior individual experience in programming.

The use of a general purpose programming language in project-based activities may also become advantageous because of the expressive amount of third party libraries, which may help the development of complex interactive books. In this level, the author is supposed to be comfortable with programming and, using the adequate third party library, may for instance create books dealing with mobile devices GPS, NFC, and so on. Authors in this level, who also want to customize piBook appearance, may also set CSS rules to modify colors, fonts and the general book layout. These features should be implemented in next versions of the tool as a pluggable third-part component.

REFERENCES

Albano, G., Iacono, U. D., and Mariotti, M. A. (2016). Argumentation in mathematics: mediation by means of digital interactive storytelling. *Form@re - Open Journal per la formazione in rete*, 16(1):105–115.

Felicia, P. (2014). *Game-based Learning: Challenges and Opportunities*. Cambridge Scholars Publishing, Newcastle upon Tyne, 1st unabridged edition edition.

Fraser, N. (2015). Ten things we've learned from Blockly. In *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*, pages 49–50.

Gobel, S., Salvatore, L., and Konrad, R. (2008). StoryTec: A digital storytelling platform for the authoring and experiencing of interactive and non-linear stories. In

International Conference on Automated solutions for Cross Media Content and Multi-channel Distribution, 2008. AXMEDIS '08, pages 103–110.

Hahn, R. (2016). Collaborative creative writing in the l2 classroom using the software Twine. In *Proceedings of the 6th Future of Education International Conference*, pages 137–142. Pixel.

Kafai, Y. B. and Burke, Q. (2014). *Connected Code: Why Children Need to Learn Programming*. MIT Press.

Krajcik, J. S. and Blumenfeld, P. C. (2006). Project-based learning. In Sawyer, R. K., editor, *The Cambridge Handbook of the Learning Sciences"*, chapter 19, pages 317–34. Cambridge University Press, Cambridge.

Leonard, L., Mokwele, T., Siebrits, A., and Stoltenkamp, J. (2016). 'Digital Natives' require basic digital literacy skills. In *The IAFOR International Conference on Technology in the Classroom*. The International Academic Forum.

Lye, S. Y. and Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior*, 41:51–61.

Menezes, H. (2012). Using digital storytelling to improve literacy skills. In *Proceedings of the International Conference on Cognition and Exploratory Learning in Digital Age - CELDA'12*, pages 299–301. International Association for the Development of the Information Society.

Prensky, M. (2009). H. sapiens digital: From digital immigrants and digital natives to digital wisdom. *Innovate: journal of online education*, 5(3):Art.1.

Quill (2017). Delta documentation. Available at: https://quilljs.com/docs/delta/. Last access: 2017-02-27.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., et al. (2009). Scratch: programming for all. *Communications of the ACM*, 52(11):60–67.

Robin, B. (2006). The educational uses of digital storytelling. In *Proceedings of SITE – Society for Information Technology & Teacher Education International Conference*, volume 2006, pages 709–716.

Wing, J. M. (2006). Computational thinking. 49(3):33–35.

Wolber, D. (2011). App inventor and real-world motivation. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 601–606. ACM.