Documentation Comes to Life in Computational Thinking Acquisition with AgentSheets

Marcelle Pereira Mota, Leonardo Serra Faria, Clarisse Sieckenius de Souza
Departamento de Informática, PUC-Rio
Rua Marquês de São Vicente 225
22451-900 Rio de Janeiro, RJ
{mmota,clarisse}@inf.puc-rio.br

ABSTRACT
This paper is about the use of live documentation in Computational Thinking Acquisition (CTA) programs with AgentSheets. AgentSheets is a visual programming environment to build games. Based on previous studies showing that semiotic relations among visual game elements could be further explored to the benefit of the learners, we designed *The PoliFacets*, a Web extension to AgentSheets cast as a live conversational document. With it, teachers and learners can follow different threads of conversation about (multiple representations of) game elements and relations between them. We present a qualitative evaluation study of *The PoliFacets* with two experienced AgentSheets instructors and three school teachers trained to coach students in CTA programs. Findings show that although our system has not yet completely fulfilled our design intent, it has led participants to gain relevant insights about their teaching and learning, as well as to articulating doubts and misunderstandings that could have otherwise gone unnoticed.

Keywords
Live documentation; computational thinking acquisition; metacommunication, end-user programming; AgentSheets.

INTRODUCTION

Live documentation has been a topic of discussion for decades. From early attempts at design rationale and critiquing systems [13][8] to task-oriented instruction wizards [17] and explanation systems [10], interactive systems that can answer questions and perform tasks in specific domains have always figured as an attractive alternative to traditional passive documentation. Progress in distributed and multimedia systems leading to investigations in collaborative knowledge building and learning environments [9] or enriched representations of content [15], have led to even more sophisticated forms of documentation and communication around it. Nevertheless, the design of efficient and effective help systems – the most common form of online documentation – has remained a challenge to this date. Work carried out only a few years ago reports that such systems are not the users' preferred choice when they need help to solve problems [16], a good candidate explanation for why online help systems typically deserve much less attention in systems and interaction design than would be expected. Using search engines to find help has in fact become common practice [23].

Documentation is a critical component in software, especially in software designed to support learning. This paper is about documentation design and use in AgentSheets [19] version 3.0, a visual programming tool to support computational thinking acquisition. In it we present *The PoliFacets*, a Web-based active documentation extension to AgentSheets.

Active documents have been conceived and defined in different ways. For example, Phelps [17] takes the "*conversation between users and a program*" as the essence of active documentation. Bicharra [4], however, takes the essence of "*active [design] documents*" to lie in the system's ability to interpret, critique and learn from observing users as they engage in various kinds of tasks. Both approaches are meant to help users *solve problems*, although Bicharra's is more explicitly committed to support *reflection in and on action* [20]. Our approach to designing *The PoliFacets* incorporates aspects of both. By using concepts from Semiotic Engineering [5], we conceived this active document as an *epistemic* tool that supports reflective activity, although it isn't able to interpret, critique and learn from users' interaction with AgentSheets. The interest of this research in the context of active documentation is thus to discuss expected and achieved gains in *reflection on* problem-solving tasks (closer to Bicharra's approach) with a system that is designed for conversation but doesn't require Artificial Intelligence apparatus (closer to Phelps's approach).

In Semiotic Engineering, interactive software is viewed as a kind of proxy for software designers and developers. Through systems interfaces, software producers have computer-mediated conversations with software consumers. Documentation in online help modules thus becomes a prime opportunity for designers and developers to ‘have a conversation with users’, not only about supported tasks and the kinds of problems that can be solved with the system, but also about the system’s value, interesting contexts of use, etc. [21]. Live documents like *The PoliFacets* are *software about software*, whose designers have twofold intent. One is to tell the users about how *it* can be used, for what purposes, when, where, by whom and why. The other, because its users are also the users of another system which constitutes the very object of live documentation, is to tell them the same kinds of things about *the other* system.

The PoliFacets was designed using the Semiotic Engineering metacommunication template [5], a carefully structured model of how to communicate *through* and *about* systems interaction. As an indication of how successful this approach to active documentation can be in practice, we report and discuss the results of a preliminary small-scale study with school teachers and AgentSheets instructors participating in a CTA project. Our focus is on how AgentSheets uses and deploys various styles of computer-mediated communication to achieve its goal.

In previous studies [6][7] we found that although AgentSheets communication is very rich, including several active documentation features that support *reflection in* (i.e., during) *action*, further semiotic engineering of game design documentation might improve the teaching-learning process in important ways. As a consequence we designed *The PoliFacets* to act as a *mediator* between AgentSheets users (teachers and students), the visual programming environment and its associated instructional resources. Mediation is achieved through descriptions, explanations and illustrations communicated as users engage in *conversations* about computer games produced with AgentSheets. The ultimate intended effect of mediation is to support *reflection on action* (i.e., about action results), by means of organizing documentation resources distributed across AgentSheets modules and deploying it as a cohesive network of conversations that can give users a better understanding of what AgentSheets *means* and what they can do with it.

The paper is structured in five sections. After this brief introduction, we present an overview of AgentSheets, emphasizing its *conversational programming* features [18], as well as its explanation and online documentation resources. Then we present *The PoliFacets* and highlight the main points in the semiotic engineering process of this tool. In a subsequent section we present the qualitative study we carried out with participants of our CTA project and report on the main findings. In the last section we discuss the meaning and implications of our current findings in view of related work. We also indicate the next steps we plan to take in this research.

AGENTSHEETS

AgentSheets [19] lets users create games and simulations through direct manipulation of objects displayed in a user-friendly visual programming interface. Simulations can be built to explore complex ideas or to communicate them to other people. By building games and simulations users can understand fundamental concepts of programming and computation, such as abstraction, algorithmic thinking, programming logic, etc.

In AgentSheets programming is centered on defining agents and their behavior. The latter is achieved by if-then rules with conditions and actions. Figure 1A shows how an agent’s behavior is visually specified. The agent’s condition to be tested is on the left side and the action to be performed when the condition is true is on the right side. If the user selects parts of the rule and presses the Explain button (bottom of Figure 1A), AgentSheets displays a brief description of how such parts are interpreted. In this example, if the user selects the upper box on the ‘if’ side of the rule and presses Explain, AgentSheets says: True if the up arrow key is currently being pressed. One of the attractive features of AgentSheets is that the textual explanation is coordinated with the visual specification of the rule using animations. For instance, the word key in the textual explanation is momentarily colorized with the same color as the rectangle ‘key’ in the visual specification shown in Figure 1A. Next the same happens with up arrow and its corresponding image, and so on. Rules are grouped into triggers like “*while the program is running*” or “*when creating a new agent*”, for example. The list of rules is tested sequentially. When all conditions for one of the rules are satisfied, the corresponding set of actions is performed, the trigger cycle is terminated (without testing subsequent rules) and a new cycle begins testing all rules in sequence again.

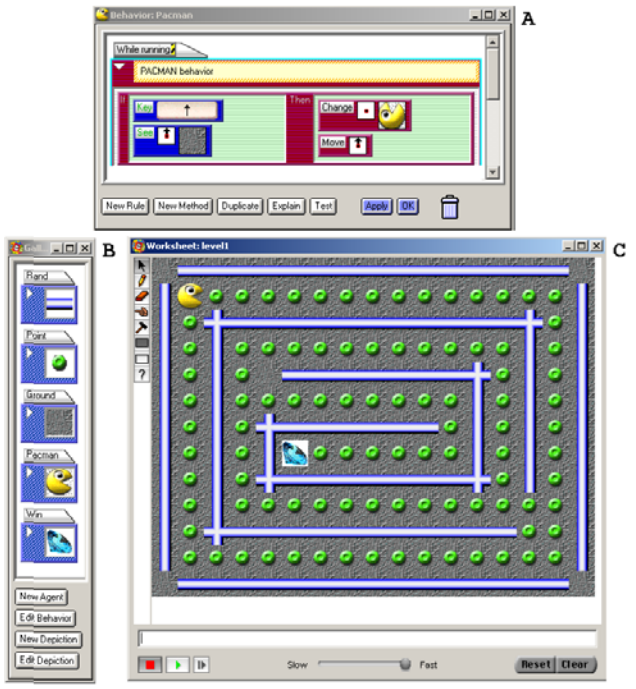


Figure 1. AgentSheets’ desktop

Figure 1B displays the gallery of agents and Figure 1C the worksheet (*i.e.*, the game board or simulation canvas) where all the agents are positioned. The whole program is built visually and agents are placed in the worksheet with drag-and-drop moves. In the gallery it is possible to create more agents and edit their behaviors. Agents can have one or more depictions. For example, the *Pacman* agent has four depictions, each one looking in a different direction, which is required for realistic visual behavior during the game. The user can run, stop or pause the game, as well as control the game speed (from slow to fast) in the worksheet. The reset button returns the worksheet to its last saved state and the clear button erases all the agents in it.

There are three types of documentation in AgentSheets: *static descriptions* (like visual programming resources and reports); *online community resources* (like wiki, game arcade, etc.); and *dynamic hints and clarifications* (mainly conversational programming features).

While a game or simulation is being executed, the user can turn on AgentSheets' conversational programming features [18]. If so, while rules are sequentially tested, a color code is used to signal that their pre-conditions are (green) or are not (red) true in that particular execution cycle. If the user runs the game at a slow speed, he can 'see' the trace of program execution on a separate window and check if the logic of the program is correct (*i.e.*, that condition-action pairs lead to the desired effects). Color and animation help users understand why agents do what they do at run time.

In an addition to conversational programming, AgentSheets provides a static HTML program report with a description of all agents and triggers. This report can be used to inspect agents' behavior and reflect on the game. However, it is not properly *communicated* in the interface. It is hiding as a single secondary option in one of the tool bar menus. The outstandingly visual nature of communication in AgentSheets makes it very unlikely that users will *get the message* about the value of this program report [6].

In a previously published study, we showed how participants of a CTA program using printed copies of a slightly extended version of the program report [7] could expand and correct prior learning. We concluded that communication of and about new game representations could support new teaching strategies and thus looked at empirical results to identify initial requirements for the development of *The PoliFacets*. At first, *The PoliFacets* is being developed by our research group to complement the AgentSheets and we decided to keep it separate to speed the studies until it is more consolidated. So it will be integrated into the AgentSheets Game Arcade. Although the project in that study being conducted in Brazil, we have a commitment to the international project, and then *The PoliFacets* is written in Brazilian Portuguese and English.

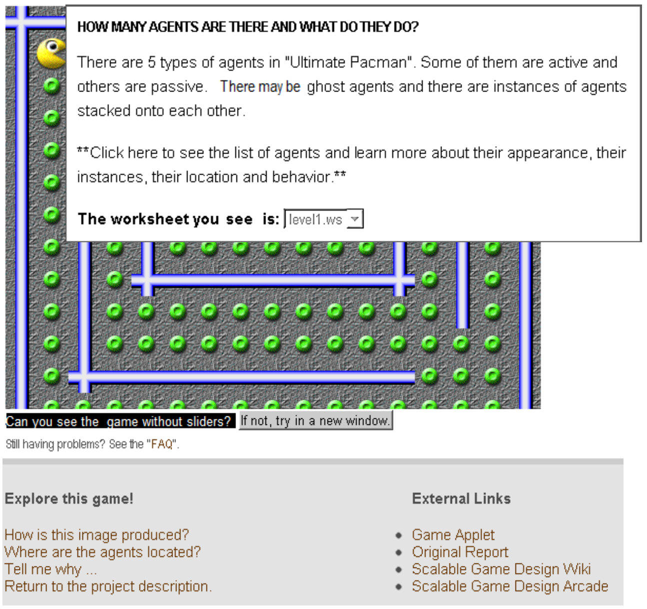
THE POLIFACETS

As already mentioned, *The PoliFacets* has been designed using Semiotic Engineering, a theory of Human-Computer Interaction where systems interfaces are seen as messages sent from designers to users [5]. The interface delivers the designer's message about how, when, where, what for and why to use systems. Users communicate directly with the system using the interface, and designers communicate indirectly with users through the system.

In this research, we introduced an *additional mediator* to enrich and explain the message AgentSheets designers send to their users through the original visual interface. Using a mediator can make the designers' message easier to understand, increasing the means and modes to explain important information whose original communication can be implicit or subject to some ambiguity. Acting as a *mediator* between (a) users, (b) AgentSheets, and (c) the various computational thinking and visual programming resources in and around this application, *The PoliFacets* actively communicates about *meanings* encoded in AgentSheets programs. Such meanings constitute the object of further conversations with the user. As a result, our system augments communication, but does not introduce new topics of conversation compared to the entire set of resources provided by AgentSheets. It is the outcome of a semiotic re-engineering of meanings, in the form of active documentation.

Technically, *The PoliFacets* is a Web application that can be invoked while interacting with AgentSheets. In it, users (typically CTA teachers and students) can explore alternative representations of their game, detect bugs, devise corrections and even see the opportunity to create new agents and behavior rules. Conversations refer to explanations, illustrations, questions & answers, contrasts, suggestions and comments about all uploaded material. Consequently users can not only see different aspects of their own projects, but also contrast their solutions with those of others. Our design intent is to allow users to engage in collaborative activity (sharing games, solutions, as well as questions and even challenges or exercises). *The PoliFacets* is thus a place for *communication, analysis and reflection*, whereas AgentSheets is mainly (but not only) as a place for *design and building*.

The generation of conversational paths starts when the user sends a project to *The PoliFacets*. The system captures and parses project information, which is then used to fill in various templates in the network of all possible conversations. Each uploaded project has its own network of conversations. There are four major sections in the network: "Agents; Worksheets; Rules", "Game Logic"; "Applet"; and "Report". Cross-section conversations can be accessed through the top menu bar (the entry page "Games", "Send New Game", "Send your Questions", "Help"). The richest part of the network of conversations is "Agents; Worksheets; Rules". As its name suggests, the topic of conversation is the essence of AgentSheets games and simulations. From this topic, a web of sub-topics unfolds. In Figure 2 we illustrate how users can talk about different sub-topics like: how many agents are in the worksheet and what they do; what the agent behavior is; where the agents are located; etc.


HOW MANY AGENTS ARE THERE AND WHAT DO THEY DO?

There are 5 types of agents in "Ultimate Pacman". Some of them are active and others are passive. There may be ghost agents and there are instances of agents stacked onto each other.

Click here to see the list of agents and learn more about their appearance, their instances, their location and behavior.

The worksheet you see is: level1.ws

Can you see the game without sliders? If not, try in a new window.

Still having problems? See the "FAQ".

Explore this game!

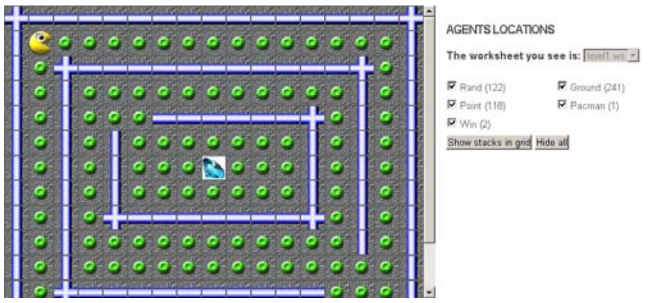
How is this image produced?
Where are the agents located?
Tell me why ...
Return to the project description.

External Links

- Game Applet
- Original Report
- Scalable Game Design Wiki
- Scalable Game Design Arcade

Figure 2. Options of conversations network

Games can have more than one worksheet and *The PoliFacets* allows for viewing alternative images. The position and behavior of agents is presented and explored in accordance with the selected worksheet. The game image can be produced from a background image with agents in the foreground or with multiple arrays of agents, often stacked on top of each other. Together produce the impression of background and foreground. If users choose the latter, the behavior of hidden agents in the background may occasionally affect the game in seemingly mysterious ways. Conversations about 'how the image is produced' point at these possibilities, complemented by conversations about 'how many agents are in the worksheet and what they do'. For example, as shown in Figure 3, the user can see the number of instances for each agent in the worksheet, as well as all the types of agents used to compose the game. *The PoliFacets* gives users an opportunity to explore hidden agents (agents whose image, or depiction, cannot be perceived when the user looks at the worksheet). When users press the Hide all button all agents disappear from the worksheet. They can then choose to see instances of agents type by type (*e.g.*, check the Win agent only to see where its two instances are located, then uncheck it and check the Pacman agent only, and so on, and so forth). In this example, the Win agent is depicted as a diamond (see the center of the image) and only one instance is visible, although *The PoliFacets* communicates that there are two instances of it in the worksheet. The other one *must* be hiding somewhere. Show and hide *conversations* will tell the answer.


AGENTS LOCATIONS

The worksheet you see is: level1.ws

[✓] Rand (122) [✓] Ground (241)
[✓] Point (118) [✓] Pacman (1)
[✓] Win (2)

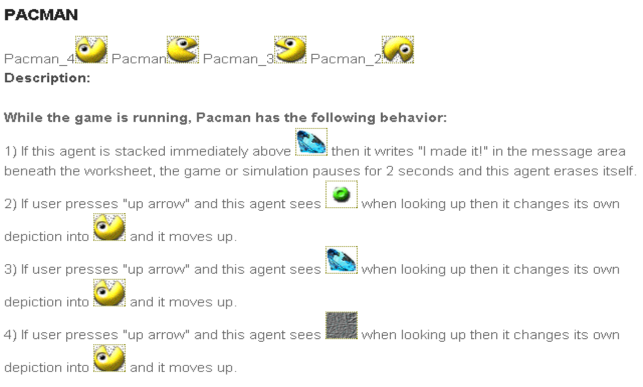
Show stacks in grid | Hide all

Figure 3. The structure of game and Agents' checklist

The structure of the worksheet plays a fundamental role in the success of game and simulation programming. Figure 3 helps to illustrate the *epistemic* effect of conversations in our live document. Our design intent in this piece of semiotic engineering is to instigate the learner to discover what is happening, starting from looking for where the second diamond is 'hiding'.

As already mentioned, hiding agents can bring up undesirable effects in the game. For example, if Pacman is programmed to *win* when it moves onto a diamond, then the hiding diamond may cause Pacman to erroneously win in some other place than the central cell of the worksheet. This case shows that our live documentation is also a powerful debugging tool.

The PoliFacets presents the logic of all agents' behavior in automatically-generated natural language text. An example is shown in Figure 4, with part of the description of agent Pacman.


PACMAN

Pacman_4 Pacman Pacman_3 Pacman_2

Description:

While the game is running, Pacman has the following behavior:

1) If this agent is stacked immediately above then it writes "I made it!" in the message area beneath the worksheet, the game or simulation pauses for 2 seconds and this agent erases itself.

2) If user presses "up arrow" and this agent sees when looking up then it changes its own depiction into and it moves up.

3) If user presses "up arrow" and this agent sees when looking up then it changes its own depiction into and it moves up.

4) If user presses "up arrow" and this agent sees when looking up then it changes its own depiction into and it moves up.

Figure 4. Example of Logic Game in natural language

We believe that the *verbal* representation of *visual* AgentSheets rules can not only boost sense-making processes that are fundamentally important for novice CTA learners, but also be used as part of different teaching strategies. For example, teachers can ask students to generate *themselves* one representation in view of the other to try and develop desirable conceptual associations. Specifically, we can explore how reflexive actions like 'erase itself' (Erase) are encoded and used in

AgentSheets, something that one of our previous studies has shown to be a problem for some students [7].

Although there are many other relevant aspects in the semiotic engineering of live documentation cast in *The PoliFacets*, we do not have the necessary space to discuss them here. Nevertheless, we believe that the examples above are sufficient to give the reader an idea of how documentation *comes to life* in the context of our research project. In the next section, the reader will see how users actually received our message through *The PoliFacets*'s interface, that is, the outcome of our semiotic engineering process.

EMPIRICAL STUDY

The main goal of this CTA project, at its current stage, is to promote computing literacy that, at later stages, enable new forms of social participation, that is, *acting through software*. Thus our first goal is to teach users how to build small programs and use it to communicate ideas in school-related activities. Teachers can apply AgentSheets games and simulations to illustrate content in their respective disciplines. Likewise, students can be stimulated to express their learning through their own versions of games and simulations.

Our empirical study was conducted within this context, as part of an ongoing CTA program carried out in a Brazilian public school since 2010.

Methodology

The goal of the study was to understand challenges in teaching and learning computational thinking and to see how Coaching Teachers, in their own training and preparation for subsequent activities with students, react to *The PoliFacets* when they first meet it. Their reaction is a sign of how our design message is received by a group of actual users. We expected results to provide at least two *sorts* of indications: firstly, how successful our semiotic engineering process was; and secondly, as a typical result of qualitative research [10], what other meanings emerge from data collected in the study, pointing at potentially unsuspected new challenges and possibilities.

We invited a group of three 'teachers to be' (T2B) in the process of training for an upcoming program with their students and two AgentSheets instructors (ASI) with previous experience. This is a purposive sampling because we are committed to supporting participants of a pioneering CTA project in Brazil, which has involved a total of three AgentSheets instructors, four coaching teachers and approximately forty students. We used questionnaires with open questions, live interviews, observations of interaction with technology and a focus group.

Participants

The T2B group included lower and middle school teachers participating in our CTA project. One is a Biology (P1) teacher and the other two are Math teachers (P2 and P3). Neither of them had any previous knowledge of programming and their training with AgentSheets took approximately 12 hours. An interesting aspect of this group is that, as they learn about AgentSheets and Computational Thinking, they are necessarily thinking about their subsequent teaching activity. Therefore, reflection in action and reflection on action [20] are both intensive.

The ASI group included two AgentSheets Instructors (P4 e P5) with expert knowledge in computing. P4 is an MSc student in Computer Science and P5 has a PhD in Human-Computer Interaction from a CS program. P4 has participated in three teacher-training programs and P5 has participated in one.

Procedure

The procedure with the T2B had four steps. We created a very simple AgentSheets variation of the well-known PACMAN game, where Pacman moves around the game space, eating food and trying to escape from monsters. In our simplified version, all Pacman does is to eat green points on its way to a blue diamond.

The first step in the procedure was to read a specifically designed test scenario and implement small changes in the game. Teachers were asked to explain what they planned to do first and then do it in AgentSheets. The second step was an introduction to *The PoliFacets*. We made a short slide presentation conveying the big picture of the system (like one finds in typical Web pages describing software products in the Internet). Then, we demonstrated *The PoliFacets* web site, navigating through its main functions. The demo included our own version of the scenario task (which did not necessarily coincide with the solutions presented by participants). We took this opportunity to emphasize that there are many ways to design games that exhibit similar perceived behavior. Through out this step, T2B could ask questions and make comments, to which we responded accordingly. The third step was asking T2B to make a second change in the game. Now they had a specific goal to achieve. They should make Pacman step onto the blue diamond. As soon as that happened, they should display a message to the user saying that he would move to the next phase, the Pacman should disappear from the worksheet and the game should terminate, in this order. The fourth step of our study procedure with T2B was a focus group. In it we had the opportunity to discuss the participants' experience with AgentSheets and their first impressions about *The PoliFacets*. During the whole procedure we collected observers' annotations, screens captures and voice recordings.

The procedure with ASI also had four steps, although different ones. In the first step ASI watched a slide presentation about *The PoliFacets.* In the second step, they accessed *The PoliFacets* web site and navigated freely to explore some of the functions presented in the slides. The third step was to answer a questionnaire with open-ended questions to collect their first thoughts about the proposed technology. The fourth step was an online chat interview for further clarification and discussion of topics explored in the questionnaire. In this second four-step procedure we gathered data questionnaires and the online chat sessions.

Results

In this section we will highlight main categories of meanings emerging from the data. We will present them merging T2B and ASI evidence because the relevant categories were found in both. Because of lack of space we will illustrate each category with only one or two pieces of evidence, although much more evidence contributed to the reported findings.

Learning Difficulties

We observed that T2B participants have difficulties to decide where to place behavior rules. They would repeatedly open and close windows before deciding where the rule should go. Moreover, they often realized that their choice was wrong and changed the rule location again. In addition to the difficulties in deciding how to define rules that refer to mutual behavior involving more than one agent, the situation was aggravated by semiotic engineering problems in AgentSheets interface, which scatters information across multiple independent windows without providing additional integrative viewing mechanisms.

P2 said: "*I will place [the rule] in the little bullet [behavior]. If see the little bullet, [then] move right (…)*". P2 did not notice that this rule formulation was wrongly associating the behavior rule to the *object* of the condition (the *little bullet*), rather than the *subject* of the action (Pacman), as required by AgentSheets rules. This fluctuation of perspective was extensively evidenced in one of our previous studies, too [6].

ASI participants talked about other common difficulties that students have in understanding the structure of game rules. They reinforced evidence such as P2's, above, saying that students have problems to realize that rules are strictly related to a single agent, that conditions and actions pertain only to the agent being currently defined. P5 said: "*During the creation of a new game, students find it difficult to 'bridge the gap' between their definition of the game and the programming [that corresponds to it]. 'Where to start' is a recurring problem*".

ASI also mentioned further difficulties with the visual encoding of game logic. All conditions appearing in the same rule block express a logic conjunction (AND). If users want to express logic disjunctions (OR) they have to create multiple rules, one for each clause in the disjunction.

P4 said: "They [Students] face difficulties in understanding that the order or rules can affect game execution. […]Initially it is difficult to understand which conditions are "additional" (AND) and which are "optional" (OR)."

P5 said: "*[Students] also experience in the programming. For example: […] when to encode rules into single x separate blocks.*"

Abstraction

ASI have talked about learners' problems with abstractions in different circumstances. P5, for example, comments that: "*Students also don't remember to associate rules with [more general computational thinking] patterns and fail to realize that behaviors can be copied from one agent to another*". This observation is actually critical in a broader CTA perspective, since abstraction is one of the fundamental skills required in computational thinking.

In the T2B group, we observed that P1 and P3 hadn't grasped the difference in meaning between classes and instances of agents. In AgentSheets this difference is ambiguously expressed by contrast between linguistic and visual representations. The 'class' is referred to by its *name*, whereas 'instances' are represented by individual *depictions* (multiple images of the agent on the worksheet). P1 and P3 defined agents' behavior by defining multiple *instance-level* rules, for each one of all the depictions of Pacman. They complained that this was a lot of work, failing to realize that they could define behavior for the whole class. P2, however, vaguely remember that this was possible and asked if there wasn't an easier way to perform the task she had to complete.

Program Execution Logic

As is apparent in P4's testimony mentioned above, understanding how the order of rules affects execution is hard. T2B learned that the order of rules was important, but they actually did not quite learn how and why order affects execution. So, as soon as something went wrong in their program, the order of rules quickly came up as a candidate explanation for error. As a consequence, changes in the order were more erratic than reasoned. P1, for instance, was confused by mutually exclusive rules. When having to define rules for Pacman behavior when moving on(to) the floor or the little green bullets he said: "*I think this will cause conflict*", failing to realize that the logic of rules was a case of disjunction and not conjunction.

This problem is actually associated with another one, which is just as difficult or more: the sequence of instructions execution in AgentSheets. Execution is carried out in rule-testing cycles for all agents. Agent behavior rules are tested sequentially until the conditions of one rule are all true. When this occurs, the corresponding action set is executed and the cycle is terminated without testing the rest of the rules defined for that particular agent. This is the main instrument to introduce and teach algorithmic thinking, but learners, as P1 above, have persistent problems with it.

Recognition of value in The PoliFacets

All participants praised the possibility of sending questions to a teacher or instructor through *The PoliFacets*'s interface. They also valued having access to Frequently Asked Questions. P5 said: "*I think this is very good. Also because the same questions come back during the lesson. The teacher can motivate the use of the FAQ […]. It will be* *much less stressful for him and the student can realize that many questions have already been answered before. Sending questions online is great if the answers come fast*".

Another praised feature of our active document was to show (reveal) the structure of the worksheet. All participants could think of actual instances when having this available would have been very useful in their learning or teaching context, especially for debugging tasks. P2 and P3 commented on different situations where they were forced to build a whole new worksheet because they simply could not spot and delete some hidden agent that was causing the game to behave in undesired ways. The amount of re-work was especially critical if they had a complex worksheet, with numerous interacting agents on it.

In AgentSheets, users must probe for hiding agents by clicking on all individual cells in the worksheet. There is no way to have like a *radiography* of the game space structure and go directly (and solely) to the spots where the origin of problems may be. In *The PoliFacets*, however, with Show/Hide stack in grid conversations combined with Show/Hide [sub-sets of agents] conversations, users can be quickly directed to problem areas (see Figure 5). This is what participants were referring to. During the presentation and demonstration of *The PoliFacets,* T2B participants quickly noticed that there were two Win agents (depicted as a blue diamond) in the worksheet instead of one, which they could see. They immediately realized it could be a problem. If the rule was the Pacman stacked somewhere above the blue diamond, the game will finish even if the blue diamond is stacked above the Ground agent.

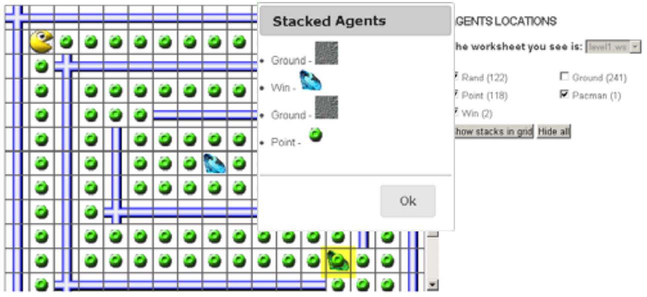


Figure 5. Structure of game worksheet hiding ground agent and showing stacked agents in highlighted point

The PoliFacets Applications

ASI participants emphasized the application of *The PoliFacets* inside and outside the classroom, as a resource both for students and teachers. P5 said: "*I think teachers would use it as an aid to increase their own knowledge and prepare their lessons*". P4 said: "*For example, a student who is having problems [with game execution] and verifies that all of the rules are implemented properly could find out that the problem is [having] stacked agents [in the worksheet]. Another student may find that a rule that he created is not associated with right agent, and so on (…) I think The PoliFacets can be an excellent tool to show students a different view of the game. Many people don't initially see what a particular problem is. But this is not because they don't know [that the problem can occur]; they don't visualize it. The PoliFacets brings another vision that can help users to find these errors and find new possibilities for the game*".

The last comment refers exactly to one of our main motivation in developing *The PoliFacets*. We intended to create new game representations that had the potential to trigger reflection and insights.

New Features

During focus groups activity P1 and P2 suggested several improvements in the structure of game worksheet. One was to name columns and rows of the worksheet grid, following the '*Battleships game board*' (as the metaphor also used in spreadsheet applications interestingly occurred to them in this context of games). Thus they would be able to have conversations about '*cell A4 or B5*'. Another improvement would be to show the worksheet grid with the total number of agents' instances on each cell. In its current implementation stage, *The PoliFacets* reveals only the total number of agents in the worksheet.

Still talking about the game structure, they further suggested that we improve the representation of stacks. ASI, in particular, suggested that we could take the opportunity to explain in more details the concept of stack, which is essential to programming.

Another important improvement suggested by P4 was this: "*The PoliFacets might also support the visualization of the relationship between agents. For example, if we see that agents exchange messages with each other, which agents [ones are talking to which] other agents on the worksheet*". More than that, this possibility would explain the critical role of some *passive* agents (*i.e.*, agents without behavior), since by simply being there, on the way of other active agents, they can influence the behavior of the latter.

P5 suggested that in addition to FAQ and sending questions to the teacher, *The PoliFacets* could provide a forum for students and teachers to discuss games, ask questions, share ideas, and so on. According to P5 there are good reasons for creating this type of resource: "*I saw […] that new questions will be incorporated in the FAQ, but I also find it interesting to provide the history of questions and answers (such as an open forum thread). It would be great if The PoliFacets allowed the teacher to make searches in forum discussions. I use this tool a lot on the Internet to find solutions to various kinds of problems. It is rare to [not find a solution to] my problem, because problems and doubts are usually recurring. Besides it is good to know I'm not the only one to have doubts about a particular topic*".

DISCUSSION

We divide our discussion in two parts. The first aims to address the findings and relating them to the features that we have (or have not) included in the current version of

The PoliFacets. In other words, this part tells how the designer-to-user message of this tool was received by users. The second part connects our findings about *The PoliFacets* with previous related work.

About the *Learning problems* associated with the difficulty in deciding where to place behavior rules, *The PoliFacets* provides automatically generated natural language descriptions of rules for all agents in the game. We hope that recurring natural language patterns showing recurring subject-action-object text will spark the learners' insight that there is regularity in this and that they should attend to it while encoding agent behavior (see text in Figure 4 and also in Figure 7 for examples). Textual representations, as was already the case with AgentSheets program report, can compensate for scattered information across multiple windows in the visual programming environment.

Regarding the interpretation of AgentSheets interface visual language, natural language text in *The PoliFacets* also provides an explicit rendition of the game logic concerning conjoined and disjoined behavior conditions. Like P4 said: "*I think the description of the rules in natural language is closer to the reality of students and more understandable*".

Moreover, learners need a case base to start making sense of what they are about to learn. If they don't have it, it's hard to get started. *The PoliFacets* is an analysis tool, which can perhaps be usefully explored at the early stages of CTA programming, exercising the conditions and consequences of selected modifications of an initial game state (as in the scenario of our experiment). Querying *The PoliFacets* and inspecting game examples can help students to *"bridge the gap"*, as one participant put it, when they have to begin to build their own game or simulation.

Developing *Abstraction* skills and mastering how this can be used in programming is fundamental to CTA. *The PoliFacets* may help the learning process if the teacher uses it to show examples of how concepts come back again and again in various documented projects. Nevertheless, we are far from being able to communicate game and programming *patterns* used in AgentSheets, in particular as they have been defined in [1]. We still need to improve considerably our natural-language generation features before we can begin to make more relevant contributions on this front.

We can, however, deal with communication simple abstraction patterns, like the difference between a class-level and depiction-level agent behavior specifications. In Figure 6, we can see an example of how AgentSheets communicates this difference by selecting agent depictions (instance-level specification) *versus* agent names (class-level specification). As mentioned in the beginning of this paper, Pacman has four depictions, which vary according with movement orientation. So, if learners don't realize that some behavior is independent of orientation, they may never make the abstraction leap from using depictions to using names in rule specifications.

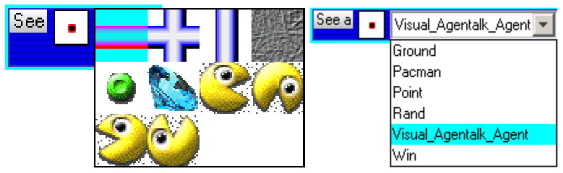
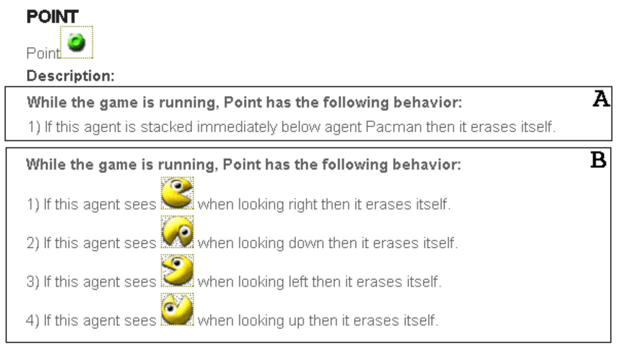


Figure 6. *See* and *See a* AgentSheets rules



POINT

Point

Description:

A

While the game is running, Point has the following behavior:

1) If this agent is stacked immediately below agent Pacman then it erases itself.

B

While the game is running, Point has the following behavior:

1) If this agent sees when looking right then it erases itself.

2) If this agent sees when looking down then it erases itself.

3) If this agent sees when looking left then it erases itself.

4) If this agent sees when looking up then it erases itself.

Figure 7. Class Vs depiction

Natural language descriptions of rules in *The PoliFacets* mark different levels of generality in rule descriptions by using iconic signs (Figure 7B) in case of instance-level specifications *versus* symbolic signs (Figure 7A) in case of class-level specifications. This semiotic regularity is designed to inferences about different levels of abstractions. However, we do not have empirical data yet to tell whether our message gets across to users or doesn't.

About the *Program Execution Logic*, the only way that *The PoliFacets* has addressed this issue is by enumerating the rules in the same order as conditions will be tested. This is however a very primitive approach, especially in view of interactions with the interpretation of logic conjunction and disjunction during rule testing. We know we have a lot of work to do in this direction, but first we must collect reliable data from learners in order to understand what interpretations they typically generate to explain game behavior. Only in view of such empirical data can we have some hope of producing explanations that will elaborate on users' meanings rather than imposing our own.

In spite of shortcomings, the gist of our design intent with *The PoliFacets* was recognized by this study's participants. They clearly saw the value of alternative and enriched representations of the game structure and conversations about it. They also seem to have grasped and enjoyed the essence of active documentation in the way we designed and implemented it.

As a complement in discussing the results of out research, we also contrast and compare *The PoliFacets* with other approaches proposed in previous related work. For example, there have been proposed strategies to address the difficulty of many students in getting started with the CTA process. One approach was to engage students in a broader interdisciplinary context. Jones and co-authors [11] explore the teaching of introductory programming with creative writing. In it pairs of students start organizing and writing a detailed story that triggers the programming process. Their research recognizes and values the creative nature of programming. In the authors' words: "*creativity first, programming second.*" This is interesting because *The PoliFacets* also resorts to writing, although to *automatic writing* (the program-generated textual descriptions of game logic) and somehow in reverse order (programming with AgentSheets first, writing second). However, as already mentioned, textual descriptions can be used in different ways, one of which being to challenge learners to produce AgentSheets programs that comply to descriptions given in *The PoliFacets*. Even if the creativity factor is not emphasized in our context yet, the semiotic infrastructure is in place for further research in this direction.

Basawapatna and co-authors [2], doing work with AgentSheets, investigated how feedback happens in the context of students using cyber learning and physical infrastructures. Their findings suggest that students preferred to give feedback verbally in-person during class than to register it virtually. In comparison, we can say that *The PoliFacets* was conceived to support activity both inside and outside the class room. Thus feedback about game projects can happen in both contexts. An interesting possibility, springing from Basawapatna's work is to trace how feedback loops occur (and possibly migrate) across the two environments.

A study by VanDeGrift [22] also explores the use of pair programming coupled with writing and talks about the advantages of this strategy. We should mention in this respect that we saw in our study some discomfort among the T2B participants with the possibility of students being able to inspect each other's games. As P1 put it: "*[they] tend to copy a lot, and then [all] you have sometimes is a copy of somebody else's game. The guy had no idea [in mind], he just looked up and started doing the same thing [as somebody else did]*". They are concerned that *The PoliFacets* might encourage copying rather than thinking. We find that this is probably because of their lack of experience with programming practices, which heavily rely on program sharing, reuse, etc. It may actually be a good thing if students look at some game representation (that is different from AS code) in *The PoliFacets* and copy it into their AgentSheets games. In particular, because of representational variations, this will develop good computational thinking skills in addition to introducing them to wide-spread program development practices. In future, it would be an interesting aspect to investigate. But, we believe that our active documentation approach somehow goes in the same directions as suggested by VanDeGrift, although we are not specifically contemplating pair-programming; we are actually contemplating group collaboration through program sharing.

Bennedsen and Caspersen [3] found that recordings of narrated programming sessions, in a kind of verbal protocol style, can be a simple, cheap and efficient way to help students improve program understanding abilities. This view was reinforced by other research by Matthíasdóttir [14]. *The PoliFacets* doesn't follow the idea of capturing verbal protocols during programming activity, but as already mentioned the semiotic infrastructure for generating augmented representations of programs can bring in certain signs that trigger the same kinds of inferences as those triggered by narratives of programming activity. The difference is, once again, the *reflection in action* compared to the *reflection on action* approach as defined by Schön [20]. *The PoliFacets* is clearly committed with the latter.

CONCLUSIONS AND FUTURE WORK

We conclude this paper with the answer we got from participants when we asked them how they would define *The PoliFacets*. All participants talked about enhanced *visibility*. T2B participants mentioned that the system lets you see all the details more closely, like a zoom. AgentSheets Instructor P5 mentioned the ability to "*see hidden details of the worksheet*" and P4 said that "*The PoliFacets is a complementary tool that aims to explain parts which are not self-explanatory*", not obvious or easily seen. These answers make us believe that we are on the right track and that *The PoliFacets* does bring about an important expansion of program-related signs from which computational thinking skills can be taught and learned.

Our goal is not to generalize results, propose methods or best practices and advices. Our scientific contribution is a different kind of help system using live documentation and it could be explored in other contexts. Our findings also showed that although our system has not yet completely fulfilled our design intent, it has led participants to gain relevant insights about their teaching and learning, as well as to articulate doubts and misunderstandings that could have otherwise gone unnoticed.

We are thus encouraged by these results, especially by T2B participants' realizations that some of the problems that had faced could have been easily solved if they had used *The PoliFacets* in the first place. We conclude that our piece of active documentation, along with its design rationale and strategy, holds the promise of being useful in CTA contexts.

After we finished this research, we implemented many of the suggested improvements in *The PoliFacets*. However, there is a lot of room for further enhancements and research. We intend to begin with further empirical studies, looking at how this tool is actually used by teachers and learners in actual CTA activity. Some of our currently targeted research questions have to do with the automatic generation of more complex verbal explanations and descriptions of the game structure and plot. We will also be working with language and writing middle school teachers to explore the possibilities of interdisciplinary educational strategies inspired by related work mentioned in the previous section. One of the long-range targets we hope to hit with this is to couple the development of computational thinking skills with that of reading and writing skills required for full-fledged functional literacy.

ACKNOWLEDGMENTS
We thank the participants of our study as well as CAPES and FAPERJ for granting financial support to this research.

REFERENCES

1. Basawapatna, A., Koh, K. H., Repenning, A., Webb D. C., Marshall, K. S. 2011. Recognizing computational thinking patterns. SIGCSE’11. ACM, USA, 245-250.
2. Basawapatna, A. R; Repenning A. 2010. Cyberspace meets brick and mortar: an investigation into how students engage in peer to peer feedback using both cyberlearning and physical infrastructures. ITiCSE’10. ACM, USA, 184-188.
3. Bennedsen, J.; Caspersen M. E. 2005. Revealing the programming process. SIGCSE ACM, USA, 186-190.
4. Bicharra, A. C. 1992. Active Design Documents: A New Approach for Supporting Documentation in Preliminary Routine Design. Ph.D. Dissertation. Stanford University, USA.
5. De Souza, C. S. 2005. The Semiotic Engineering of human-computer interaction. Cambridge, Mass.: The MIT Press. “Omitted for blind refereeing”
6. De Souza, C. S., Garcia, A. C. B., Slaviero, C., Pinto, H.,; Repenning, A. 2011. Semiotic traces of computational thinking acquisition. EUD’11, Berlin, 155-170. “Omitted for blind refereeing”
7. Ferreira, J. J., De Souza, C. S., Salgado, L. C. C., Slaviero, C., Leitão, C. F., Moreira F. 2012. Combining cognitive, semiotic and discourse analysis to explore the power of notations in visual programming. VL/HCC’12. Submission accepted. “Omitted for blind refereeing”
8. Fischer, G.; Lemke, A. C.; Mastaglio, T. & Morch, A. I. 1990. Using critics to empower users. CHI’90. ACM, 1990, 337-347.
9. Fischer, G. 1999. Symmetry of igorance, social creativity, and meta-design. CC’99, ACM, 116-123.
10. Johnson, W. L.; Erdem, A. 1997. Interactive Explanation of Software Systems. Automated Software Engineering. Springer Netherlands, 4, 53-75.
11. Jones, M. E., Kisthardt, M.; Cooper, M. 2011. Interdisciplinary teaching: introductory programming via creative writing. SIGCSE’11. ACM, USA, 523-528.
12. Lazar, J., Feng, J. H., Hochheiser, H. 2010. Research methods in human-computer interaction. New York: Wiley.
13. MacLean, A.; Young, R. M. & Moran, T. P. 1989. Design rationale: the argument behind the artifact. CHI’89. ACM, 247-252.
14. Matthíasdóttir, Á. and Geirsson, H. J. 2011. The novice problem in computer science. CompSysTech’11. ACM, USA, 570-576.
15. Maybury, M. T. 1994. Knowledge-based multimedia: The future of expert systems and multimedia, Expert Systems with Applications, Volume 7, Issue 3, 387-396.
16. Novick, D. G., Elizalde E., and Bean N. 2007. Toward a more accurate view of when and how people seek help with computer applications. SIGDOC’07. ACM, USA, 95-102.
17. Phelps, L. 1997. Active documentation: wizards as a medium for meeting user needs Proceedings of the 15th annual international conference on Computer documentation, ACM, 1997, 207-210.
18. Repenning, A. 2011. Conversational programming in action. Visual Languages and Human-Centric Computing. IEEE Symposium, pp.263-264, 18-22.
19. Repenning, A.; Ioannidou, A. 2004. Agent-Based End-User Development. Communications of the ACM. Vol. 47, 43-46.
20. Schön, D. A. (1983). The reflective practitioner: How professional think in action. New York: Basic Books.
21. Silveira, M. S., De Souza, C. S., Barbosa, S. D. J. 2001. Semiotic engineering contributions for designing online help systems. In Proceedings of the 19th annual international conference on Computer documentation. ACM, USA, 31-38. “Omitted for blind refereeing”
22. VanDeGrift, T. 2004. Coupling pair programming and writing: learning about students' perceptions and processes. SIGCSE’04. ACM, 2-6.
23. Welty, C. J. 2011. Usage of and satisfaction with online help vs. search engines for aid in software use. SIGDOC’11. ACM, New York, USA, 203-2