

Semiotic traces of computational thinking acquisition

Clarisse Sieckenius de Souza, Ana Cristina Bicharra Garcia, Cleyton Slaviero,
Higor Pinto, Alexander Repenning

Departamento de Informática, PUC-Rio
Rua Marquês de São Vicente 225, Rio de Janeiro – RJ, Brazil
clarisse@inf.puc-rio.br

Instituto de Computação, UFF
Rua Passo da Pátria 156, Niterói – RJ, Brazil
bicharra@ic.uff.br; cslaviero@gmail.com; higor@bcc.ic.uff.br

Computer Science Department, University of Colorado at Boulder
430 UCB Boulder, CO 80309-0430 - USA
ralex@cs.colorado.edu

Abstract. Computational thinking involves many different abilities, including being able to represent real and imaginary worlds in highly constrained computer languages. These typically support very selective kinds of perspectives, abstractions and articulation compared to the unlimited possibilities provided by natural languages. This paper reports findings from a qualitative empirical study with novice programmers, carried out with AgentSheets in a Brazilian public school. The driving research question was: How do meanings expressed in natural language narratives relate to computational constructs expressed in programs produced by novices? We used semiotic and linguistic analysis to compare meaning representations in natural and artificial texts (game descriptions in Brazilian Portuguese and Visual AgenTalk code). We looked for recurring relations and what they might mean in the context of computational thinking education. Our findings suggest that the semiotic richness of AgentSheets can be explored to introduce different aspects of computational thinking in principled and theoretically-informed ways.

Keywords: Computational thinking education; End user programming languages; Semiotic analysis; Discourse analysis; AgentSheets

1 Introduction

A prime requirement for end user development to succeed is that end users be able to think computationally. Among other things, end users must have the ability to express *what they mean* in computable form, that is, to build representations of real (or imaginary) objects and phenomena using constructs of highly constrained computer

2 **Clarisse Sieckenius de Souza, Ana Cristina Bicharra Garcia,** Cleyton Slaviero, Higor Pinto, Alexander Repenning

languages [12]. These typically support only very selective kinds of perspectives, abstractions and articulation compared to natural languages, which support the expression of virtually anything that speakers can conceive of.

For novices, incremental elaboration of computer representations usually starts from imprecise mental representations that can be expressed in equally imprecise natural language discourse. When this sort of discourse is externalized, it creates tangible instances of signs that support subsequent semiotic transformations until formal and precise expressions of meaning can be used to compose computable code fragments. These fragments of artificial code blend together with natural signs and expand the semiotic universe of the novice programmers, helping them to compose larger structures of representations which eventually constitute a meaningful and executable computer program.

In this paper we explore the connection between natural language representations of program meanings and their corresponding computational encoding. We discuss findings from a qualitative empirical study with novice programmers, carried out during a Scalable Game Design project [18] with AgentSheets [16] in a Brazilian public school. We used semiotic and linguistic analysis to compare meaning representations in game descriptions expressed in Brazilian Portuguese and their corresponding program version in AgentSheets. We looked for recurring relations in the data collected from a group of 9th graders that had no previous training in computer programming. Our goal was to find what these relations might mean in the context of computational thinking education. As is the case with all qualitative research, our findings are not predictions about novice behavior. They point to three factors that may play a role in computational thinking pedagogy using agent- and object-oriented programming paradigms: entity naming strategies; token/type relations; and transitive structure changes when contrasting natural language and computer program representations.

The paper is structured in five sections. After this brief introduction, we present the essential details of the empirical study we did with a group of Brazilian students. Then we describe the method of analysis that we applied to the data. Next, we report our findings, illustrating the kinds of evidence that support them. In the last section we discuss our findings in the light of related work and indicate future directions that we want to explore.

2 A Study with Brazilian Students

We worked with a group of twenty 9th-grade students, thirteen females and seven males, between 14 and 16 years of age. They volunteered to participate in a short Scalable Game Design program [18] after class. The school is located in Niterói, 14 km away from Rio de Janeiro, across the Guanabara Bay. It is a public school whose teachers are associated with Universidade Federal Fluminense, a large public university. Most of the students in this school come from low-income communities in surrounding areas. Our group was led by a Geography teacher, also a volunteer, who had been using computers and GIS applications in his teaching.

Semiotic traces of computational thinking acquisition 3

	'Which 3 disciplines you like most?'			'Which discipline you dislike most?'		
	MALES	FEMALES	ALL	MALE	FEMALES	ALL
PORTUGUESE	0%	54%	35%	29%	15%	20%
MATH	86%	31%	50%	0%	46%	30%
SCIENCE	43%	62%	55%	14%	8%	10%
PHYS ED	71%	69%	70%	0%	8%	5%
ENGLISH	14%	46%	35%	0%	8%	5%
GEOGRAPHY	86%	31%	50%	0%	0%	0%

Table 1. Participants' most liked and disliked disciplines in the curriculum

In **Table 1** we show the most liked and disliked disciplines in this group's opinion. Their preference for outdoor activities is very clear (70% said they liked Phys Ed classes). Notice the trace of gender characteristics with respect to disciplines involving languages (see preferences for Portuguese and English) and Math (see like vs. dislike distributions).

	3 most frequent activities when using computers		
	MALES	FEMALES	ALL
GAMES	100%	38%	60%
CHAT	57%	46%	50%
SOCIAL NETWORKS	86%	85%	85%
NEWS	14%	23%	20%
SCHOOLWORK	14%	85%	60%

Table 2. Most frequent activities of participants when using computers

In **Table 2** gender characteristics stand out once more. All of the boys listed games as one of the most frequent activities when using computers, whereas just about 1/3 of the girls said so. However, almost all girls listed schoolwork as a frequent activity, against a very low percentage of boys who said so. Connecting with people in social networks was unanimously appreciated by all participants.

The project was an introduction to computational thinking with AgentSheets. It consisted of seven 2-hour weekly sessions. The goal of the program was to teach students to design a simple game related to environmental issues. For example, students built games where the player had to chase wildlife hunters, prevent forest fires, collect polluting garbage, etc. The teacher learned how to program with AgentSheets and then taught the students. Members of our research team provided technical help when needed.

The first four sessions in the project were an introduction to AgentSheets and computational thinking. Students played with AgentSheets *Frogger*, a simple version of the famous arcade game originally developed by Sega. By inspecting agent behavior in *Frogger*, the students learned the basics of Visual AgenTalk, AgentSheets programming language. They also had the opportunity to learn basic computational

4 **Clarisse Sieckenius de Souza, Ana Cristina Bicharra Garcia,** Cleyton Slaviero, Higor Pinto, Alexander Repenning

thinking concepts used in AgentSheets simulations, like collision, absorption, transportation, generation, diffusion, polling/counting and user input commands.

The purpose of the study was to know how these novice teenage programmers used AgentSheets expressive resources (agents' names, depictions, behavior-defining rules, etc.) to encode what they *meant* to do with their games. We aimed to achieve an in-depth understanding of the interpretive and expressive processes involved in program codification. Following a qualitative methodology, we analyzed the collected data using a combination of linguistic, semiotic and cognitive perspectives (see section 3).

At the beginning of the program, participants were asked to fill out a questionnaire with general information about themselves. All seven sessions, which took place in the school's computer lab, were videotaped. Two observers took notes and produced a short narrative of what happened in each session. In the last three sessions, we interviewed students and teacher. Most of the results reported and discussed in this paper come from these interviews and the corresponding versions of the game.

Of the twenty students who volunteered to participate, three never came to the sessions. We used data from ten out of the remaining seventeen (seven females and three males). This was mainly because some students missed the final sessions for different reasons, and because hardware problems with computers damaged critical files of two projects.

3 Method of Analysis

Our method of analysis is structured as a semiotic triangulation. The pivotal concept used in the process is that of a sign. Different semiotic theories have different definitions for it. We use Peirce's definition and sign classification system [15]. A sign is anything that some particular mind (typically a human mind) takes to stand for something else in some particular respect or context. There are three important elements in the structure of a sign: the representation (called 'representamen' in Peirce's theory), which stands for something else; the object (that which the representation stands for); and finally its assigned meaning (called 'interpretant'). Unlike other triadic theories of meaning [13], Peirce's is open-ended in the sense that the 'interpretant' is itself another sign (whose interpretation is yet another sign, and so on). This infinite digression introduces a process perspective on meaning, which can be easily traced in everyday life. It says that our interpretation of signs like words, images, scent, gestures, situations, is constantly evolving, influenced by previous meanings. The unlimited recursive process is halted and resumed for pragmatic reasons. A lack of resources like time, knowledge, motivation, interest, can cause evolutionary interpretation to stop until more resources are encountered.

Peircean Semiotics is thus especially fit to analyze how our group of students interpreted AgentSheets and gradually acquired computational concepts. Since the theory postulates meaning as evolution, what we will discuss here is a window on interpretation. The results should be taken as indications of semiotic processes that are in place, and can possibly be facilitated with new teaching strategies, new tool features, etc. They are not predictions of what will necessarily happen in similar

Semiotic traces of computational thinking acquisition 5

groups and situations. The value of our research lies in revealing new aspects and perspectives in the phenomena we observed.

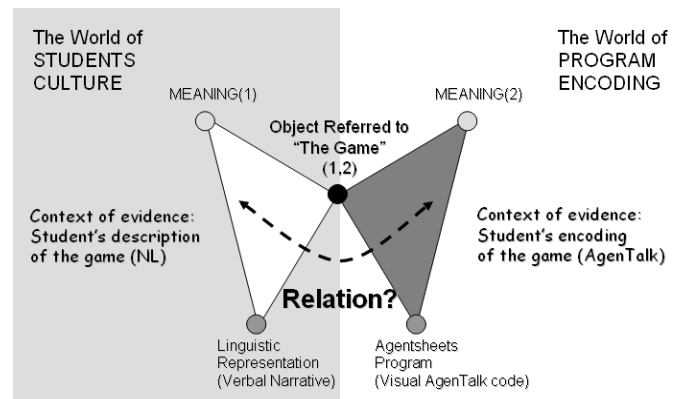


Fig. 1. A visual sketch of the triangulation process used in this research

The triangulation process is visually sketched in **Fig. 1**. On the left-hand side, against the shaded background, are signs that belong to the students' reality, the world of culture from which we factor out signs related to AgentSheets interface, programming language and game experience. These are represented on the right-hand side of the image. The two triangles represent sign structures, with three vertices: representation (bottom); object (center); and meaning (top). Note that triangles share their sign *object*, which is borrowed from the world of culture and used in the world of programming. The shared object is the *game* itself, a culturally-embedded computer artifact, which is mentally conceived (cognitive stance), verbally described (linguistic stance), and programmed (computational stance) in Visual AgenTalk.

To achieve this triangulation, we proceed with the following steps:

1. **A linguistic analysis of the verbal description of the game produced by students during interviews.** This analysis is based on narrative discourse, and identifies typical elements of narrative structure: characters (protagonist and antagonists); plot (temporal and causal relations among actions carried out by characters); setting (resources and obstacles); and goal (winning and end of game conditions). We paid particular attention to nouns and verbs used in verbal descriptions, and to the emerging semantic categories and relations that they indicated.
2. **A semiotic analysis of the corresponding game structure (agents' names, depictions and behavior), followed by a comparative analysis of the game execution (a player's control and perception of agent behavior during the game).** The game structure indicated abstractions and conceptual modeling constructs that the participants elaborated in the process of building the game. It also indicated their encoding choices and strategies to produce desired visual effects perceived during the game play. The comparative analysis between game structure and game execution gave us insights on how the participants realized the

6 **Clarisse Sieckenius de Souza, Ana Cristina Bicharra Garcia,** Cleyton Slaviero, Higor Pinto, Alexander Repenning

representational mediation between the game as an artifact and the meanings that populated their semiotic universe. For example, when the player sees “the hunter kill the monkey”, the game structure may actually specify that “the monkey disappears when it sees the hunter”. This case of inverted transitivity can only be noticed and investigated if we compare program execution and structure.

3. **A contrastive analysis of relations between meanings derived from an interpretation of verbal descriptions and an interpretation of program structure and execution.** This analysis is based on the correspondence between nouns, verbs and semantic relations in verbal narratives, on the one hand, and agents’ names, depictions, behavior and effect during simulation, on the other.
4. **The final integration and interpretation of results.** In the context of computational thinking education, we focused on the semiotic resources made available and salient in AgentSheets. Our aim at this stage is to produce a cohesive characterization of how this particular group of students *makes sense* of the whole computational thinking activity supported by AgentSheets. We also want to know what this sense-making process may *mean* for researchers interested in this and related topics.

4 Findings from our Study

The ten projects showed different levels of complexity in computational thinking terms. Likewise, verbal descriptions of the game showed different levels of elaboration and verbal fluency. However, one did not go with the other. There were very cohesive and efficient verbal narratives, whose computer implementation was very rudimentary, and vice-versa. For example, Participant 2, a boy that could barely produce an understandable description of his game in Portuguese, produced a highly sophisticated program for a two-phase game, with more than 20 agents, and elaborate spatial constraints. In Figure 2, we show a translated excerpt of his verbal narrative (see left side) and a visually annotated snapshot of his game setting, showing different agents, with causal and temporal connections between them (see right side).

Notice that Participant 2 constantly switches pronouns in the verbal description (I, he, you). Notice also the frequency of use of a semantically vacuous noun (‘guy’) and deictic references (‘here, here, and here’). Trees, meadow, grass, river, bridge – none of these visual elements present in the computer narrative were mentioned in the verbal description of the game. Neither were the bicycle and the fact that the hero of the game could only take one of the two one-way bridges to cross the river. Participant 2 had difficulties to express himself verbally all through the interviews, but he was absolutely *fluent* using another semiotic system and medium, the computer.

Many of the projects had a relatively large number of ‘passive’ agents (agents without behavior) that were not deployed in the game setting. This is an indication that participants who did this probably spent considerable time preparing the setting of the game, but did not get to a stage of putting all pieces together into a rich game plot. In most cases, these passive agents were not explicitly mentioned in the verbal description produced by participants. Moreover, nouns semantically associated with

Semiotic traces of computational thinking acquisition 7

classes or types of entities like ‘animals’ and ‘garbage’, and with what we might refer to as a *semantic field* in itself, like ‘the forest’, were frequently encoded as a collection of agents. We found evidence like: raccoons, monkeys, parrots, rabbits for ‘animals’; and plastic bags, cans, and other kinds of litter for ‘garbage’. Different kinds of trees, meadows, flowers, trails, lakes and rivers often stood for ‘forest’. A large number of agents, however, posed a harder programming challenge in the game. Defining behavior rules for each of the ‘active agents’, which should be at least moving around during the game, if not interacting with one another or with the player, was not only a time-consuming but also a conceptually harder task.

"..I begin with this guy, then I have to kill the hunters that are here and here. Then after he kills the last hunter, he becomes this other guy here and he gets an armor. Then he has to kill the lumberjacks that are here, here, and here. [...] The goal is to kill the hunters, free the animals, and after you kill the hunters, you become someone else, because you get the armor. [Participant 2]



Fig. 2. Participant 2's game versions in verbal description and computer program

During this initial phase of the project, students did not have the appropriate programming abstractions to avoid duplication of code. For example, when programming an agent's encounter with another agent, students had to define four rules (*see up, down, left, and right*), or even eight rules (to include diagonal adjacency). Consequently, in many games there was not much action going on during the game. When action was more intense on screen, this was typically the result of replicas of a particular agent, whose behavior was to ‘move randomly’ in specific areas of the game space. It was thus interesting but not surprising to see that the diversity and richness of meanings associated with the game setting were encoded mainly in static form, in contrast with a predominantly naïve encoding of diversity and richness in terms of behavior.

The analysis of verbal descriptions also showed that, although all games had a protagonist (the agent controlled by the player), not all of them had antagonists. For example, one of the girls described her game like this:

I will scatter all this garbage in the forest. [The monkey] will have to collect it, and clean the forest. (Participant 5)

Notice that although there is a *goal* to be achieved (to collect all the garbage), there is no clear definition of where this garbage is coming from and how fast. Moreover, there is no definition for a winning or gaining condition. The *moral* of this story is to

8 **Clarisse Sieckenius de Souza, Ana Cristina Bicharra Garcia,** Cleyton Slaviero, Higor Pinto, Alexander Repenning



educate the players, showing that cleaning the forest is a *tedious* job. In a previous passage, Participant 5 said:

In real life, it is not the monkey, but people that collect the garbage, and we should be aware that they are working [*sic*], of all work that they have to do. There are disgustingly messy people around [...] They are not aware that their mess and garbage can damage Nature. (Participant 5)


Notice that this participant did not master the game *genre* as a way to communicate her message. However, the tediousness of having to clean-up other people's mess was clearly conveyed, as a *model representation* of what the participant had in mind. Therefore, we can see in her program the trace of computational thinking acquisition even though the game as such is not enjoyable.

The lack of antagonists was not necessarily a sign of tediousness. Some games had obstacles – passive agents that constrained the action of the protagonist by just *being there*. A number of participants explored the idea of a *labyrinth* in the game. The protagonist had to find his way out of it, avoiding obstacles that simply did not let the player go through. They did not destroy the player, but made him or her lose time, for example. Actually, the construction of a visual sign for a labyrinth or trail, led us to a recurring category in computer sign-making activities among this group. In AgentSheets, agents can have various *depictions* associated to them. In *Frogger*, for example, the 'frog' has two depictions, which can be associated to two different *states* of the agent: frog (normal state, apt to play); and squished frog (dead after colliding with a truck, unable to continue playing).

Different depictions can lead to useful abstractions in AgentSheets programming. For example, there are two related conditions – *stacked* and *stacked-a* – that operate at different levels of abstraction. So, if there are two depictions for the agent BRIDGE:

'bridge'  and 'west_side_bridge' ,

rule 1, below, defines behavior for agent 1 when it is stacked on any agent with depiction 'west_side_bridge'. Rule 2 defines the behavior of agent 2 when stacked on any bridge, regardless of how it is depicted.

1. If I am stacked-immediately-above an agent that looks like this , then I [...]
2. If I am stacked-immediately-above a BRIDGE, then I [...]

This feature introduces a fundamental concept in computational thinking, the distinction between types and tokens, in spite of some naming constraints that may lead to equivocal interpretations. When creating a new agent, the user must give it a name and primary depiction (*e. g.* 'bridge'). Although depictions can be freely changed and added after the agent is created, its *name* cannot be changed and, by necessity, it is associated to the first depiction assigned to it (the *default* depiction). Therefore, although 'animal', for example, is an appropriate 'type sign' to refer to 'raccoon(s)', 'monkey(s)', and other *animals*, once a new agent is named 'animal', the default 'animal' depiction has to be created, leading to potential breakdowns in systematic token/type relations between agent depictions and agent names. In Figure

Semiotic traces of computational thinking acquisition 9

3, we show a comparison of object names and depictions in *Frogger*. On the left-hand side, we see a snapshot of the current version of AgentSheets Gallery of agents. On the right-hand side we show the sketch of an alternative interface design, where names and default depictions do not have to share the same identifier.

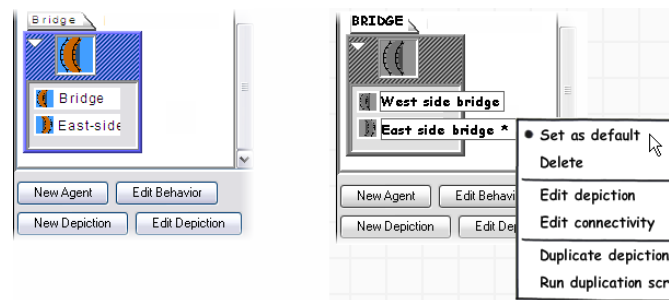


Fig. 3. Different interface alternatives to represent agent depictions in AgentSheets

In one case (left side of Figure 3), a reading of the interface leads to a breakdown with 'is-a' relationships. Although a 'west-side-bridge' *is a* 'bridge', saying that a 'bridge' *is a* 'bridge' challenges our interpretation. In the other (right side of Figure 3), the choice of an agent's name and default depiction being independent, token/type relations lend themselves to a more consistent reading (an 'east-side-bridge' *is a* 'bridge' and a 'west-side-bridge' *is a* 'bridge', regardless of which one is chosen as a default depiction).

Data collected in our study clearly indicates that many students had the right intuition about type/token relations, and tried to encode it in the program. For example, the strategy used to delimit the area where the protagonist could move was often one of creating a physical boundary around it with a linguistic marker for *type*. One girl created four agents and deployed them at the edges of the game area: "top-barrier", "bottom-barrier", "left-barrier" and "right-barrier" (Participant 9). A boy surrounded the game area with four agents: "tree 1", "tree 2", "tree 3" and "tree 4". The same strategy was used by another girl (Participant 10), who created "fence 1", "fence 2", "fence 3" and "fence 4". Depictions were the same or nearly the same, showing an important meaning invariant across related agent representations.

This is an important factor to consider in computational thinking acquisition for at least two reasons. First, the presence of linguistic markers of *type signs* in students' naming strategies suggest that they were ready to deal with a higher level of abstraction in visual and textual representations of the game than the programming tools made available to them in the project allowed for. Second, a nearly opposite strategy was also found. Some students used a single agent (with single depiction) to represent the boundaries of the game area. However, interesting distinctions in behavior when the protagonist reached the edge of the space were not correctly encoded. For example, Participant 4 had difficulty to program the behavior of the player agent when encountering "leaves". She put leaves on three of the four sides of the game space. The fourth side had a different configuration – a patch of leaves separated the agent from the water. The behavior of the agent when encountering boundary leaves should be to *back up*, and when encountering the leaves next to the

10 Clarisse Sieckenius de Souza, Ana Cristina Bicharra Garcia, Cleyton Slaviero, Higor Pinto, Alexander Repenning

water should be to step on them and proceed to the water. This created a conflict in rules, and the final behavior of the agent was determined by rule order. However, the agent clearly did not behave as desired and got *stuck* when reaching certain positions on the game grid.

Another finding regarding the triangular relationship between semantic concept (signified by words appearing in verbal descriptions of the game), agent name, and agent depiction had to do with replicated agents. As mentioned above, we found cases of one-to-one and one-to-“linguistically-related many” depiction/name relations in the data. Let us call these *token-indicator* signs and *type-indicator* signs, respectively. An examination of the former shows that distinct semiotic processes were in place. In some cases, like the specification of the protagonist agent, a token-level sign had a unique instantiation in the game. This happened in all 10 games we analyzed. In other cases, token-level signs had many *replicas* instantiated in the game. For example, in most games there were agents like ‘monkey’ or ‘flower’, whose names did not so clearly represent a *class* or *type*, like those with a linguistic type marker, especially in view of its *meaning* communicated by the game setting. See Figure 4, for example, where Participant 9 represents her game setting.

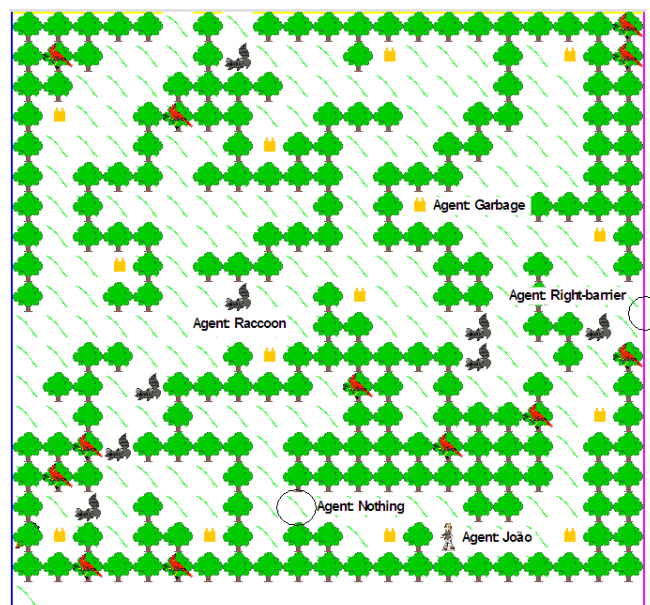


Fig. 4. Game setting visual representation with replicated agents

There are many *replicas* of raccoons and garbage on the ground, and a single instance of “João” (a proper name in Portuguese), the protagonist. Semantically, each raccoon instance is a different individual (token) of a class of animals we call ‘raccoon’. Had the game been a *model* of the animal world, the behavior of each individual raccoon should follow its own particular programming, and not be the same as is the case with *replicas*. This distinction is not important in the case of



Semiotic traces of computational thinking acquisition 11

linguistically marked type signs like the already mentioned agents named as “top-barrier”, “bottom-barrier”, “left-barrier” and “right-barrier”. Although the four *barriers* are composed of aligned *replicas* of the corresponding barrier agent (the right barrier is actually an array of 20 instances of “right-barrier” agent depictions), our interpretation of the visual sign integrates all instances of barrier into a single conceptual unit, identified by its semantic role (a boundary). So, we actually *expect* that all behaviors will be the same, which reinforces the interpreted meaning assigned to the unified object. There is also a wonderful semiotic choice in the game code that generates the setting shown in Figure 4. The *name* of the agent depicted as a diagonal line, replicated to form the *path* in the forest where the player can navigate and collect plastic bags, is “nothing” (‘nada’ in Portuguese). This is an indication of how Participant 9 conceptualized the game – the protagonist can walk in places where there is *nothing* to impede it. However, ‘nothing’ must be visible, must have a depiction, whose appearance and name really do not matter. This shows that this participant had an intuition of formal representation principles involved in computational thinking (*i. e.* that computer objects must be linguistically signified).

Another semiotic process that Participant 9 (among others) gives us a chance to illustrate is that the computer codification and the verbal account of the same referent object (see Figure 1) prompt for signs of very different nature. Her verbal description mentioned a labyrinth (“The goal of the game [is] to go through this labyrinth.”), which is not directly encoded as an agent – active or passive – but is clearly shown when the program is executed. There are various dead ends in the path that “João” can tread as the game evolves. So the labyrinth is signified by depictions and behavior of many other agents in the game, but it is never *named*. This is a complex programming skill that must be acquired and mastered if we want students to be able to build more sophisticated computational models of various kinds of phenomena – real or imaginary.

The last finding we present in this section is related to semantic transitivity. When analyzing verbal descriptions of the game, we saw that, in general, students produced evidence of *naturalistic* transitivity. That is, when talking about causal relations involved in agent interactions, most of the time they used preferential transitive structures in Brazilian Portuguese to describe what happened (*e. g.* “he collects the garbage” or “he kills the hunter”). In some cases, they also used intransitive structures that left causal relations partially undefined (*e. g.* “he meets the ranger and dies” instead of “he meets the ranger and is killed” or, alternative, “he meets the ranger, and the ranger kills him”). Pragmatic implicature in the text made it easy to recover the causal chain in this case. The most probable reason why the agent died when meeting the ranger was that *the ranger did something to it* that caused its death, and not that it committed suicide in the sequence of the meeting, for example. Nevertheless, in all but one case, students built games with at least one occurrence of causal relations *programmed* in reverse order.

Inverted transitivity is present in *Frogger*, the game that was used in the first four sessions of the project, and illustrates a computational thinking pattern called diffusion [1]. For example, the following behavior rules are defined for the FROG:

- 1 If STACKED (immediately above , ) Then SAY (Cannot walk over a turtle maker. That's cheating!), and ERASE () , and STOP-SIMULATION ()

12 **Clarisse Sieckenius de Souza, Ana Cristina Bicharra Garcia,** Cleyton Slaviero, Higor Pinto, Alexander Repenning

```
2  If STACKED (immediately above , █) Then SAY ( I cannot swim! ), and
    ERASE ( █ ), and STOP-SIMULATION ( )
```

Line 1 above is saying that if the frog is on top of (the depiction of) the turtle maker, then: the frog says, “Cannot walk over a turtle maker. That’s cheating!”; erases the object in the same location where it stands (*i. e.* the frog erases itself); and stops the simulation. Line 2 says that if the frog is on top of (the depiction of) the river (a patch of blue color), then: the frog says “I cannot swim!”; erases the object in the same location where it stands (*i. e.* the frog erases itself); and stops the simulation.

The presence of these signs in the program encoding gives us the opportunity to appreciate aspects of *code patterns and reuse*, a prime feature in object-oriented programming. The frog erasing itself and stopping the simulations lends itself to the interpretation that this is a *suicidal move*, and makes perfect sense in the context of the game. The frog is the semantic agent of all actions in line 2: *it* says that it cannot swim, *it* disappears, and *it* causes the game to stop. Now, when looking at line 1 it seems semantically inconsistent to suppose that the frog will commit suicide by stepping on an island (the surprising depiction of a “turtle maker”), or that it will reprimand itself for cheating. Maybe the island should magically eliminate the unknowing frog and stop the game after shouting ‘You cannot walk over a turtle maker. That’s cheating!’. However, although this would probably be more consistent with the fantastic logic of the game, it would have the programming cost of encoding a new kind of rule for *turtle makers*: if stacked immediately *below* the frog, then say [something], erase stacked object at location (frog) and stop the simulation. Another cost would be the effect of diffusion in this case: to transfer the encoding of frog behavior to other agents, preventing the programmer from seeing all threats and opportunities associated with the frog in the same chunk of code.

Students in our group showed us numerous cases of inverted semantic transitivity of actions. We give two illustrations. Participant 1 encoded behavior stating that when the player saw the ranger, the player erased itself and stopped simulation. Participant 8, who followed the Space Invaders model in his game, had a canon shooting water in the sky to destroy flying lanterns that fell on trees and caused big fires. His encoding, however, transferred transitivity from the shooting to the lanterns, and defined that if the lantern met water, then it erased itself. Although in this case one could argue that a switch in perspective might, indeed, allow us to describe the action from the lantern point of view, there is no way to select one perspective and keep with it throughout the whole process. Switching from one to the other is perfectly possible and, as noted before, potentially advantageous for program maintenance. We must ask ourselves if there are cognitive issues associated with this when it comes to building consistent computational models. Which representation must be taken as a *model* of a given phenomenon: sign structures in program encoding or the visual effects they produce?

5 Discussion and Future Work

Our work is related in different ways to different kinds of previous research. It is connected to previous educational projects for teaching computational thinking not

Semiotic traces of computational thinking acquisition 13

only with AgentSheets [18], but also with other technologies such as Alice [8, 3] and Scratch [9, 19], for example. The innovative aspects of our research come mainly from the semiotic approach adopted in this study, but also from the context of our research. As far as we know, this is the first computational thinking project carried out with AgentSheets in a Brazilian public school.

Our work inherits from all previous work with AgentSheets a particular emphasis on the acquisition of computational thinking, rather than programming skills. Thus, we do not focus on how students in our group used programming elements such as variables, loops, conditionals, and the like. Instead, we analyze signs, representations and meanings, tracing the semiotic thread leading from the students' psychological and cultural world into the world of computing. This creates a connection with previous research on cognitive dimensions of notational systems. For example, our analysis of token/type relations is directly related to Blackwell and Green's [2] abstraction and resistance to change (viscosity) dimensions. As mentioned in the preceding section, there is a cost in programming behavior rules for similar but not identical agents (*e.g.* fence and barrier agents surrounding the game space). If the students had more readily available abstraction mechanisms (which are provided by AgentSheets, but require more sophisticated computational thinking skills), maybe they would have spent less time encoding repetitive agent behavior. Cases of premature commitment were evidenced by the fact that agents' names cannot be changed in AgentSheets, which led to inconsistent relations between agents' names and depictions. Two additional cognitive dimensions were observed in the data: visibility and hidden dependencies. For example, in our discussion of inverted transitivity we remarked that although the visual effects of the underlying program might suggest that Agent A's action caused effect on Agent B, the encoded behavioral rules might actually state that Agent B was acting in a particular way in the presence of Agent A. In the textual specification of the program, the relation between A and B is thus invisible when we look at Agent A's behavior. There is a hidden dependency with Agent B.

There is no prescription for representing agents' behavior in encapsulated form (following the traditional object-oriented programming paradigm) or diffused form (crosscutting various agents). As Repenning has previously shown, there are computational advantages in programming AgentSheets simulations in exactly the opposite way as prescribed by OOP [17]. He calls it 'programming with antiobjects', and discusses efficiency gains if, for example, when programming collaborative behavior of agents on a platform, interactions among them are encoded as *platform behavior* rules, rather than the other agents' behavior rules. The complexity of program can be impressively reduced. The point we make in our analysis is that we should have a clear notion of which aspect(s) of computational thinking we want to explore – the achieved simulation or the underlying rules and structures that define it? Which one stands for a model of the phenomenon it refers to in culture, nature, or an individual's imagination? Which one is a *sign* of the phenomenon?

According to the semiotic theory we adopt, both are. Moreover, one stands for the other, which supports the triangulations we used in our analysis. This semiotic perspective somehow *dissolves* the tension discussed by Frasca [6], that narratives are not an appropriate model for games. He claims that simulations are a much better model because they are *active* representations, and not *passive* text. Juul [7] raises a

14 **Clarisse Sieckenius de Souza, Ana Cristina Bicharra Garcia,** Cleyton Slaviero, Higor Pinto, Alexander Repenning

similar point, although he does not contend the narrative model for games. He remarks that whereas narratives take a time perspective that is typically oriented towards the past (literally speaking, one cannot *narrate* the future), games are oriented towards the future. They unfold into the future as the players interact with it. There are interesting ontological switches in this perspective if we conjecture that the reason why we do not *narrate* the future is that we cannot *predict* it. Actually, when the future *is predictable*, narratives like “I will open my hand, drop the mirror and it will break into pieces on the floor” are completely acceptable. Since computer games are formally programmed, they become considerably predictable. Thus, a narrative into the future is not as constraining as Frasca believes. However, if the narrative is rooted in the formal specification of the game, that is, in the linguistic stance (rules of behavior and agent configurations) then inverted transitivity may again turn into an issue.

The essence of the above argument has been captured by Mor and Noss [10], who speak of programming as mathematical narrative. Their research explores how narratives can be used to construct mathematical knowledge and encode it formally, in an environment that resists ambiguity. Computers are an excellent semiotic medium to explore different perspectives on representations, while keeping them always within the limits of precision and formality. Can narratives be used as scaffolds to acquire formal knowledge?

In a way, Myer’s work on natural programming [11, 14] is a positive answer to the question. Natural language is not formal and precise, but it is the richest means of expression that we have. The various kinds of mechanisms that can be used to control focus, switch perspectives, refer directly and indirectly to real and imaginary things, elicit verbal behavior from others, trigger semantic associations – all of these and more support and develop human mental activity. Thus, it is not surprising to see that programming languages actually incorporate – even if partially – a number of these mechanisms in their designs. Should or could AgentSheets increase its semiotic power by, for example, supporting aspect-oriented programming [5]? This might create interesting new possibilities to deal with inverted transitivity in novice and advanced programming projects. A separation of concerns and perspectives might for example help us keep with direct transitivity in a domain model (logic) perspective, while allowing for diffusion in a program architecture (software engineering) perspective. The two perspectives depend on related but not identical computational thinking skills.

This aspect perspective on programming leads us to our final topic of discussion, which is related to research work on using direct manipulation interfaces to teach Euclidean geometry. Sedig and co-authors [20] report interesting results with three alternative styles of direct manipulation interfaces for a Tangram educational game. With the first, smaller Tangram polygons can be freely dragged, dropped and rotated to fill a larger polygonal area with arbitrary geometric form. The acquired skills at this stage have to do with spatial reasoning, but the learners develop a model that is not consistent with Euclidean Geometry. For instance, they believe that translation and rotation are just physically constrained move and turn operations, with no formal parameters. With the second interface style, learners cannot manipulate polygons directly; they must operate on specific visual controls representing axes, angles,

Semiotic traces of computational thinking acquisition 15

vertices, and so on. Ghost images show a preview of the effects achieved by control manipulations. As a result, at this stage, learners begin to think of (and learn) geometric concepts involved in translation and rotation. Yet, they can rely on preview images to verify the meaning of manipulations against desired effects. The third style of direct manipulation eliminates the ghost image and requires that the learner be able to preview mentally (*i. e.* to calculate) the effects of geometric variable and operation control manipulations. This leads to mental representations in problem solving procedures that resemble an algebraic formulation of geometric functions. It is only at this stage, according to Sedig and co-authors, that learners really understand Euclidean geometry concepts that can be used resolve Tangram challenges.

De Souza and Sedig [4] have shown that there are powerful semiotic mechanisms in place at each stage of abstraction in the geometric game. They claim that the different styles of direct manipulation actually correspond to different types of signification systems. Increasing levels of formality associated with the meaning of visual objects actually correspond to a progression in semiotic categories that evolve from the perceptual (first style), to the associative (second style) and then to the formal (third style) aspects of a geometric *model* representation.

The same applies to our study with AgentSheets. At the introductory stage, the goal of the teacher is to bring students in contact with computational thinking, stressing the perceptual qualities of the game execution. Then, along the pedagogical path, we should gradually incorporate and explore associative and formal qualities in the expression of computational models produced by the students. This would help them acquire increasingly complex representational skills and multiple perspectives on computation.

Our next steps in this research go exactly in this direction. We want to take advantage of the semiotic richness of AgentSheets and explore alternative teaching strategies, in order to learn how computer signification systems can be better learned and used by middle school students. The fact that we are using a semiotic perspective will also allow us to track potential cultural differences between Brazilian and non-Brazilian students participating in Scalable Game Design projects in various countries.

Acknowledgments

We thank the director, teachers and students of COLUNI for much appreciated collaboration in this project. We also thank CNPq, FAPERJ, Petrobras and NSF for supporting different parts of this research project.

References

1. Basawapatna, A.R., Koh, K.H., and Repenning, A. Using scalable game design to teach computer science from middle school to graduate school. In *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education (ITiCSE '10)*. ACM, New York, 224-228 (2010)
2. Blackwell, A.F. and Green, T.R.G. Notational systems - the Cognitive Dimensions of Notations framework. In J.M. Carroll (Ed.) *HCI Models, Theories and Frameworks*:

16 **Clarisse Sieckenius de Souza, Ana Cristina Bicharra Garcia,** Cleyton Slaviero, Higor Pinto, Alexander Repenning

- Toward a multidisciplinary science.* San Francisco: Morgan Kaufmann, 103-134. (2003)
3. Cooper, S., Dann, W., and Pausch, R. Teaching objects-first in introductory computer science. In *Proceedings of the 34th SIGCSE technical symposium on Computer science education (SIGCSE '03)*. ACM, New York, 191-195. (2003)
 4. de Souza, C. S. and Sedig, K. Semiotic considerations on direct concept manipulation as a distinct interface style for learnware. In *IHC2001 - IV Workshop de Fatores Humanos em Sistemas Computacionais*. Porto Alegre: SBC. 229-241 (2001)
 5. Elrad, T., Filman, R.E. and Bader, A. Aspect-Oriented Programming *Communications of the ACM* Vol. 44(10) 28-32 (2001)
 6. Frasca, G. Simulation versus narrative: introduction to ludology. In M.J.P. Wolf and B. Perron (Eds.) *The Video Game Theory Reader*. London: Routledge. 221-235 (2003)
 7. Juul, J. Games telling Stories? A brief note on games and narratives. *Game Studies*. Vol. 1(1). Online at <http://www.gamestudies.org/0101/juul-gts/> (2001)
 8. Kelleher, C. and Pausch, R. Using storytelling to motivate programming. *Communications of the ACM*. Vol. 50(7) 58-64 (2007)
 9. Maloney, J.H., Peppler, K., Kafai, Y., Resnick, M., and Rusk, N. Programming by choice: urban youth learning programming with scratch. *SIGCSE Bulletin*. Vol. 40(1) 367-371. (2008)
 10. Mor, Y. and Noss, R. Programming as mathematical narrative. *Int. J. Continuing Engineering Education and Life-Long Learning*. Vol. 18(2) 214-233. (2008)
 11. Myers, B.A., Pane, J.F., and Ko, A. Natural programming languages and environments. *Communications of the ACM* 47(9) 47-52. (2004)
 12. National Research Council Committee for the Workshops on Computational Thinking (2010) *Report of a Workshop on The Scope and Nature of Computational Thinking*. Online at <http://www.nap.edu/catalog/12840.html>
 13. Ogden, C. K. and Richards, I. A. *The meaning of meaning*. (8th edition) New York, NY. Harcourt, Brace & World, Inc. (1989)
 14. Pane, J.F., Ratanamahatana, C.A., and Myers, B.A. Studying the language and structure in non-programmers' solutions to programming problems. *Int. J. Human-Computer Studies* Vol. 54(2) 237-264 (2001)
 15. Peirce, C. S. (1992, 1998) *The Essential Peirce, Selected Philosophical Writings*, Volumes 1,2. Edited by Nathan Houser and Christian J. W. Kloesel. Bloomington, IN. Indiana University Press (1992, 1998)
 16. Repenning, A. and Ioannidou, A. Agent-Based End-User Development. *Communications of the ACM*. Vol. 47(9) 43-46 (2004)
 17. Repenning, A. Collaborative diffusion: programming antiobjects. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications (OOPSLA '06)*. ACM, New York, 574-585
 18. Repenning, A., Webb, D., and Ioannidou, A. Scalable game design and the development of a checklist for getting computational thinking into public schools. In *Proceedings of the 41st ACM technical symposium on Computer science education (SIGCSE '10)*. ACM, New York, 265-269 (2010)
 19. Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., and Kafai, Y. Scratch: programming for all. *Communications of the ACM*. Vol. 52(11), 60-67 (2009)
 20. Sedig, K., Klawe, M., and Westrom, M. The Role of Interface Manipulation Style and Scaffolding on Cognition and Concept Learning in Learnware. *ACM Transactions on Computer-Human Interaction*. Vol. 8(1), 34-59 (2001)