

COLLABORATIVE COMPUTATION ON THE FLOOR

Y. FERNAEUS & J. THOLANDER

DSV, Stockholm University, Sweden

ylva@dsv.su.se, jakobth@dsv.su.se

Abstract. One common problem in children's programming is that it is difficult to understand the relation between program creation and program execution, partly because computation is "hidden" in several respects while the program is being run. To address these problems we are developing role-playing activities to support children's understanding of computational processes. We describe one such activity where children enact the roles of computational rules that control the visual elements in a video game. The most important aspect of this activity was that it gave the children a first person experience of the relationship between autonomous agents and how they together formed the behaviour of a working system. Other issues that were brought to light were when and why rules are triggered and the effects that rearrangements of behaviour and objects had upon the system as a whole. In a user scenario, with one computer and one user, these issues are difficult to study in such a concrete way.

1. PROGRAMMING AND TANGIBLE COMPUTING

Tangible computing focuses on finding ways to bring computation out into our physical environments. A pragmatic reason for this is that more and more of our everyday environment gets embedded with computational properties and we need to find ways to make the interaction with these devices efficient. A more philosophically oriented reason for this is the turn towards conceptualising cognition as an embodied and situated phenomenon (Lakoff and Johnson 1999). In order to make use of such a conception of cognition, ways to more closely integrate physical and social reality in interaction with software has been called for (Dourish 2001).

We are conducting a project where children aged 8-12 use the animated programming system ToonTalk (Kahn 1996) to design and program their own animated computer programs, ranging from simple games and virtual ecosystems. When children make animated systems in ToonTalk, they work with graphical elements onto which several small autonomous agents, called robots, are attached. A robot may control various aspects of the picture's behaviour on the screen, for instance how it moves around and what will happen if it collides with another picture.

Even though programming often involves a number of collaborative activities such as testing and debugging, most programming tools, professional as well as educational, are designed with the individual user in mind. Furthermore, since programming is difficult to learn, there are a number of potential benefits in designing programming tools with properties that more naturally lend themselves to collaborative activities. Therefore we are currently designing learning activities that allow children to collaboratively program and execute games and simulations.

One common problem that children have in learning to program is to understand the relation between program creation and program execution (Repenning and Perrone 2000) and (Smith and Cypher 1999). In concurrent programming languages, like ToonTalk, this might become especially problematic since it is impossible to follow the triggering of each rule as they execute in parallel. Moreover, in games and simulations implemented in ToonTalk robots run on the "back" of pictures and the computation is therefore "hidden" in several respects. Another conceptual difficulty is that systems, like those that our students develop, can be described at multiple levels:

Algorithms → robots → visible objects in the game → the game as a whole

To understand how a game or a simulation works, one needs to grasp the difference between these levels and how they are related. Programming in such a system includes the definition of objects and the relation between objects, as well as algorithmic aspects of conditions and actions. The focus of our work is on the structure of programs and the relation between program elements, rather than on specific computational algorithms. This is partly because our previous work (Tholander, Kahn et al. 2002) has shown that those aspects were the most feasible for the targeted age groups to work with, and partly because most programming systems for children are rule-oriented (Smith and Cypher 1999), (Repenning and Perrone 2000), (Klopfer and Um 2002). Therefore, we believe that a lot could be gained by considering higher-level aspects of program construction, where the syntactical details of rules are less important.

2. ROBOT ROLE-PLAY

In line with research on “participatory simulations”, as described by Resnick and Wilensky (1998) and Colella (2000), we are developing role-playing activities for learning about computational processes. Within the participatory simulation scheme, groups of individuals use their bodies to enact elements of a computational simulation by following a set of simple rules. An example could be that participants represent virtual ants, and then observing patterns that arises from the group as a whole. By letting students enact the computational processes themselves, they get new perspectives on the relation between programming and program execution.

However, if each participant only represents objects that are visible in the computational simulation, several of the complexities of the computation remain hidden, such as the triggering of a particular action as an effect of an interaction between two objects. To overcome this we are designing role-playing activities where participants enact the roles of the independent sub-behaviours of an object. This is supposed to make computational aspects salient and concrete at levels below the level of objects, in order to make the execution of behaviours and rule-oriented aspects of computation enactable.

2.2. *An example activity*

In the following we describe one of the activities that we have developed. All activities start out from simple games that we run on a computer. In this case we used a simple game consisting of three objects and as many robots as there were children (8 in our case) that controlled the objects. We started the activity by playing the game together on a big screen and discussed the different objects, their respective behaviours, and any other programming rules in the game. The children were asked to make suggestions about how the different behaviours in the game and how they believed those controlled the different objects. We wrote down all suggestions on a whiteboard and discussed them until we found a limited set that we could agree upon. The next step of the activity was to move from execution on the screen to execution on the floor. We used the following resources for this: magnified paper copies of all the moving elements in the game, a large mat, made of nine A3-papers taped together (to serve as background), large paper representations of interaction devices (the arrow keys), paper labels describing the workings of each behaviour, or robot, in the game.

Each child was assigned responsibility for the execution of one rule in the game, as describes on the paper labels that we had prepared. The enlarged paper game elements were then laid out on the mat and everyone arranged themselves so they could reach the elements that they were about to control (see Figure 1). The player character was controlled by one of us standing on either of the enlarged arrow keys. Each time any of the children's rules triggered they were to perform the correct action to their objects.

We played the game by iteratively evaluating all the rules in the system. In each iteration the children should carry the actions if the conditions for their behaviour were fulfilled. However, the setup not only allowed us to execute the game, but we could also reprogram the game by changing the object that somebody was assigned to so that they would carry out the same actions but for a

different object. For instance, by moving the behaviour “remove player when right leaf touches the player” from one of the leaves to the turtle, the game was changed from being about avoiding the falling leaves to instead being about trying to catch the leaves.

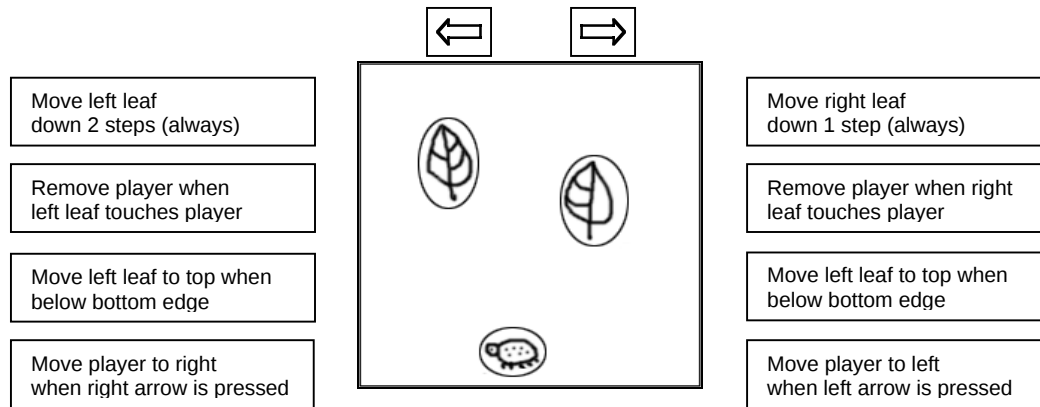


Figure 1. The game structure and the robot rules, as arranged on the floor

As an informal and collaborative group exercise, the activity turned out to be valuable in letting the children externalise their ideas and collaboratively think about computation and game design. The most important aspect of this activity was that it gave the children a first person experience of the relationship between autonomous agents and how they together formed the behaviour of a working system. Other issues that were brought to light were when and why rules trigger and the effects rearrangements of behaviours and objects had upon the system as a whole. In a user scenario with one computer and one user these issues are difficult to study in such a concrete way.

3. FUTURE WORK – TANGIBLE ROBOT ROLE PLAY

In the setting described above, there is no feedback in the system regarding the triggering or the execution of the rules, so it was all up to us as teachers and the students themselves to recognize whether any of the participants made actions that did not correspond to how the robot was programmed. Even if this did not cause us any major problems, we believe that it would be a valuable aid if we could extend the system by including some computational properties into it. By giving each object a sensor that communicates with the software on the computer, we will be able not only let students enact the running of the program, but also in a more actual sense reprogram the game off the computer and then study its effects in the real game on the screen.

4. REFERENCES

- Colella, V. (2000). Participatory simulations: building collaborative understanding through immersive dynamic modeling. *The journal of the learning sciences* 9(4): 471-500.
- Dourish, P. (2001). *Where the Action Is - the foundations of Embodied Interaction*. Cambridge, MIT.
- Kahn, K. (1996). ToonTalk - An animated programming environment for children. *Journal of Visual Languages and Computing* 7(2): 197-217.
- Klopfer, E. and T. Um (2002). *Young Adventurers - modelling of complex dynamic systems with elementary and middle school students*. ICLS, Seattle.
- Lakoff, G. and M. Johnson (1999). *Philosophy in the Flesh*. New York, Basic Books.
- Repenning, A. and C. Perrone (2000). Programming by Analogous Examples. *Your Wish is My Command: Giving Users the Power to Instruct their Software*. H. Lieberman. Cambridge, MA, USA, Morgan Kaufmann: 351-370.
- Resnick, M. and U. Wilensky (1998). Diving into complexity: developing probabilistic decentralized thinking through role-playing activities. *The Journal of the Learning Sciences* 7(2): 153.
- Smith, D. C. and A. Cypher (1999). Making Programming Easier for Children. *The Design of Children's Technology*. A. Druin, Morgan Kaufmann Publishers: 202-221.
- Tholander, J., K. Kahn, et al. (2002). *Real Programming of an Adventure Game by an 8 year old*. ICLS, Seattle.