Real Programming of an Adventure Game by an 8 year old

Jakob Tholander
Stockholm University
97 PUBLICATIONS **1,665** CITATIONS
SEE PROFILE

Ken Kahn
67 PUBLICATIONS **1,031** CITATIONS
SEE PROFILE

Real Programming of an Adventure Game by an 8 year old

Jakob Tholander, Ken Kahn, and Carl Gustaf Jansson
Department of Computer and Systems Sciences, Stockholm University, KTH, Forum 100, 164 40 Kista, Sweden
Tel: + 46 8 674 74 79, Fax: + 46 8 703 90 25
Email: jakobth@dsv.su.se

Abstract: We analyze episodes from a case study involving an eight-year old boy's use of a programming environment for computer game development, especially designed for young children. We carefully examined the quite substantial changes he made to an adventure game in collaboration with a researcher. From analysis of his activities we identified two issues that were crucial to his success and we discuss their relation to design of programming environments for children. First, the concrete interface metaphor makes it possible to refer to program language elements with simple indexical terms and through pointing. Second, the adult tutor provided support that allowed the boy to focus on the issues central to the task at hand. However, the locally situated nature of these two processes also limits the possibility of developing a general vocabulary which can be used in different contexts and situations.

Introduction

An analysis of episodes from a case study involving an eight-year old boy's use of a programming environment for computer game development, especially designed for young children are presented. We carefully examined the quite substantial changes he made to an adventure game in collaboration with a researcher. From analysis of his activities we identified two issues that are crucial to his success and we discuss their relation to design of construction tools for children. The analysis primarily concerned the programming concepts that he worked with ranging from behavior composition to data structure manipulation to higher-order programming. We also analyzed the interactions between the boy and the researcher that supported him. We focused on how the interactive support from the tutor helped the boy to move between narrative descriptions, and underlying computational structures and processes. We also discuss higher order processes involving design considerations and analysis of program mechanisms. The results suggest that the tools he used were critical to his success, but could also limit the possibility of developing a general vocabulary for referring to programming concepts. The case study was conducted within the context of the Playground project, where children between 6 and 8 years were engaged in after school game programming activities. We present a number of episodes and discuss them in terms of programming and in terms of interaction between the boy and the researcher. A similar approach can be found in Roschelle's (1996) analysis of two girls' interaction with a physics simulation.

We intend to show how real programming can be carried out by young children in ways that are meaningful to them, and which includes other important activities than automated interface manipulations. We use the word "real" in order to emphasize that the boy built real computer programs through meaningful interaction with program language elements.

Novice programmers are known to have a wide range of different problems when learning to program, e.g. predicting program behavior and debugging. Several of these become especially important if the programmers are young children. Our results suggest that several of these problems can be overcome through adequately designed scaffolding in the form of software as well as human tutors. However, programming for children is seldom a learning goal in itself, instead it is often used to develop skills in other areas such as mathematics (Kafai, 1995), science, and technology (diSessa, 2000). This requires that children develop skills that let them refer to contexts outside of the local situation in which they are currently acting, for instance through theoretical programming concepts. Studies of how, and if, programming skills do transfer to other domains and to problem solving in general are inconclusive in many respects (Palumbi, 1990), (Clements & Meredith, 1992). For instance, in Rader, Brand, & Lewis' (1997) studies of children's understanding of program rules in KidSim it was concluded that children seldom explore or learn about the underlying programming model of the simulations they construct. Our study are relevant for these results, indicating that even though children are able to construct quite advanced programs, the language and actions which they learn to use to achieve this are too a large extent constructed locally and therefore not easily used in other contexts and situations.

Jonathan and the ToonTalk Playground

The following episodes involve the work of Jonathan (not his real name), called J below, working with a researcher, called R. He used a game-making platform designed and built in ToonTalk (TT) including game components, sample games, tools, and pre-built behaviors. All of the elements are open for inspection and editing. TT is a complete programming language that resembles a computer game. Rather than rely upon text or pictures to convey programs, in TT, programs are built in an animated virtual world. Programs are constructed by training robots to manipulate tangible program elements such as birds and trucks (see Figure 1). Robots can be combined into teams to form more complex operations. In order to use ToonTalk to build video games, graphical pictures are used to represent characters and objects in the game. In ToonTalk pictures are controlled through sensors and remote controls, such as screen position, speed or size. There are also general sensors for detecting mouse movements, mouse clicks, and other system events. ToonTalk robots can be placed on the backside of pictures (flipping over) to operate on its sensors and remote controls. For the user to manipulate the environment a virtual hand is used to control a number of different tools: a magic wand for copying objects, a vacuum cleaner for erasing and removing things (called sucking and spitting), a bicycle pump for changing the size of objects, and notebooks are used to store objects.

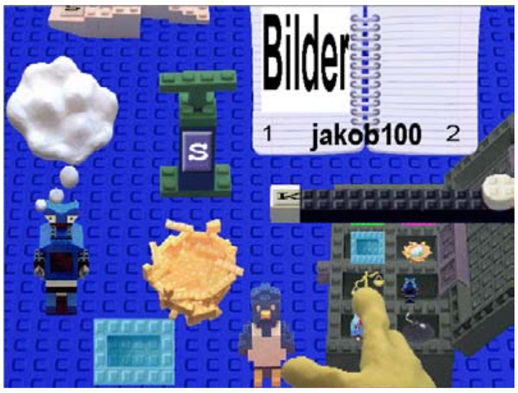

Figure 1: The ToonTalk programming environment

J used an adventure game where a player character could be moved around different scenes connected through portals. Each scene had bad-guy characters and useful tools that could be picked up. The goal of the game was to find a treasure that was hidden in one of the scenes. In order to get there you had to pass obstacles, fight enemies, etc (See Figure 2).

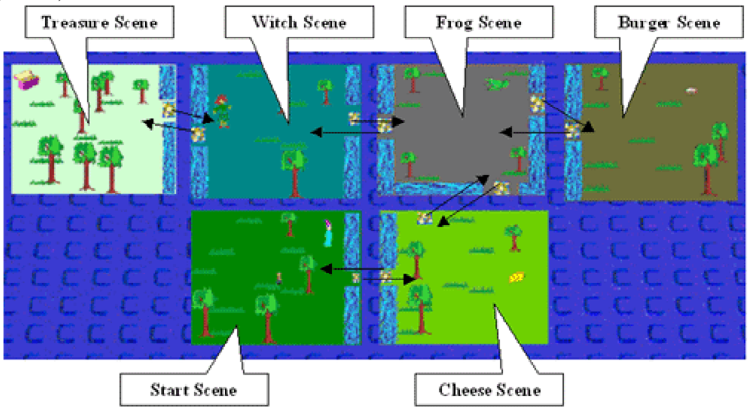

Figure 2: The spatial layout of the adventure game

The functionality of the game consists of a game engine where the scenes, and how the player character is transferred between the different scenes through portals, are managed. The game also includes non-player characters and game tools that are composed from pre-built behaviours such as “I move left and right”, “I blow up objects that I hit”.

During the first session J worked together with his friend Nick. They had worked together with the project once a week during the previous three semesters and had become the two most skilled children in the school, which they were aware of and quite proud of. Data is presented in the form of verbal transcripts, and descriptions of gestures like pointing described in brackets. In some transcripts screenshots illustrating the involved program code is also shown.

Part 1: Constructing an explanation of how a compound behavior was constructed.

Nick had worked with the adventure game alone at a previous session and had changed a magic cheese to not only transfer a behavior for moving vertically to the player character (at the start the player character could only move horizontally), but to also give the player character the behavior to blow up things that it collided with. Nick was proud of the new behavior which he made sure J knew. J was impressed by Nick’s story and wanted to take a closer look at how this was constructed. We started discussing how the magic cheese originally worked, and then continued to think about how Nick could have used it to construct the explosion behavior.

J could not explain how Nick constructed his magic cheese by simply using his conceptual knowledge of TT and Playground. However, as he starts to unpack the original game apart he is able to gradually build a conception of the workings of the magic cheese and how Nick had constructed his new behavior (see Figure 3).

1. *Researcher - Do you remember what happened when you came to the cheese?*
2. ***J - Yes, you could move upwards as well***
3. *R - Something special right, before you got there you couldn't --*
4. ***J - =you couldn't move up and down***
5. *R - Then when you got to the cheese you could!*
6. ***[J sucks the cheese out and flips it over and R reads the text label to him.]***

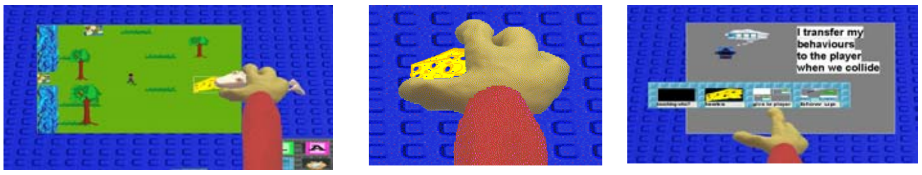

Figure 3: Sucking out the magic cheese , picking it up, and flipping it over to investigate how it was programmed.

7. *R - "I transfer my behaviors to the player when we collide"*
8. *R - Take it out and look in that box*
9. ***[J takes out and looks at the behavior from the box]***

In this excerpt R guides J to start thinking about the properties of the magic cheese that concerns the mechanisms that makes it run. In order to help J to start breaking down the behavior of the magic cheese into appropriate sub-behaviors R asks J what happened when you came to the cheese (turn 1). This leads J to take out the cheese to make an in-depth investigation of how it was constructed. R continues to guide J and to narrow his focus of attention on the mechanism that Nick had used to create his magic cheese.

Through joint efforts J and R have decomposed the game so that J can start to construct an explanation for how Nick built his magic cheese. Even though most of J's activities were non-verbal he was not led through the software without understanding what he was doing. But in order to start understanding the behavior it is necessary for him to also interact with the program elements that implements it, seen e.g. in turn 6 where he takes the cheese out and flips it over without being instructed to do so in any way. Hence, J's knowledge of the programming environment is at this point tied to the specific actions involved in interacting with the program elements. It is therefore difficult for him to speak about the objects without practical interaction with the objects.

Constructing the explanation

In the previous section R helped J to find out about how the robots controlling the magic cheese worked. Below R explains the details of that behaviour.

9. ***[J takes out the box and looks at it]***
10. *R - Yes, look at that, so what did the cheese robots do?*
11. ***J - Well, so you could move up and down***
12. *R - And it is the robots here that make you get those. The robots in there [points to a hole in the robot's box with the behavior in (Figure 4)]. So you could put any robots you like in there.*
13. *R - Do you want me to show you?*
14. ***J - Yes***
15. *R - The robot in here. What he does is that when he collides with someone he gives these robots in this box to the one he collides with. So, now he gave those robots.*
16. *R - So, can you guess what Nick did?*
17. ***J - He programmed a robot so that it would make everything that it hit explode. And then he put it in there [points to the third hole in the box where the behavior that is transferred is kept]***

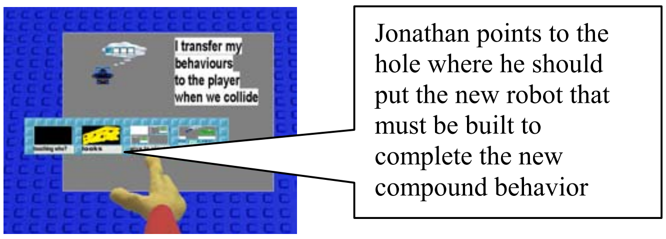
I transfer my behaviours to the player when we collide
Jonathan points to the hole where he should put the new robot that must be built to complete the new compound behavior

Figure 4: Jonathan points as he constructs and explanation for how Nick has built a new behavior

18. *R - Right, and then he put it in there [points to the third hole in the robot's box, which is the hole where transferred behaviors are kept]*

R gives J an elaborated explanation (turns 12 and 15) for how the robot on the backside of the cheese transfers a behavior to any object that the cheese collides with. In turn 16 R asks J how he thinks that Nick constructed his magic cheese behavior and J now rather precisely describes the different programming actions needed for that to be achieved. In turn 17 he explains how one first needs to program a new robot that blows up anything that it hits. Then he continues by explaining and showing that the robot should be combined with the robot that transfers a behavior to anything that it collides with and points to the hole in the robot's box where it should be added (Figure 4).

The programming elements play an important role in J's explanation. Initially, it is difficult for him to construct a hypothesis regarding how this behavior was constructed purely from his conceptual knowledge of Playground and TT. He seems to lack some of the conceptual and language based resources which are needed in order to construct such an explanation by himself. However, the highly visual environment and the structure of the program elements, replaces these conceptual resources. J is able to construct his explanation by actually decomposing the programming code so that the different sub-elements of a particular behavior can be inspected (Figure 4). Furthermore, in his explanation refer to the code by pointing to a hole in the robot's box, instead of having to describe the elements in computer science terminology.

Another important aspect shown in this excerpt is how J distinguishes between talking about the game as it is played out on the screen, and talking about the programming mechanism that make the game run the way it does. J's focuses on the robots, their boxes, and descriptions of these in terms of program behavior. Furthermore, R actively supports J to shift from game level to program level descriptions when he in turn 10 asks J what the *cheese robots* do, and not only, like previously, what the *cheese* did. Hence, indicating that it is appropriate to consider the game from a programming perspective. Initially, J basically replies to this with a similar description as in the previous excerpt (the cheese makes you move up and down) (turns2 above, and 11). However, later he qualifies his explanation by explicitly referring to robots as well as programming, and also by pointing (17) to the code where changes had to be made.

Part 2: Constructing and predicting a new behavior

In the following, J programmed a behavior similar to the one he constructed an explanation for in the previous section. This involved a number of specific aspects of programming, such as decomposition of behaviors into sub-behaviors, extensions of existing program code, and transfer of programming concepts to new contexts. It illustrates how he with appropriate support from R is able to go from an idea expressed mostly in narrative terms, to a running program implementation.

19. *R - So, do you think we should change this so that you get another behaviour or do you think it's fine with only getting up and down*
20. ***J - No, it's not good!***
21. ***J - That when you. Well, you are adding a robot. The robot should be programmed so that when you get to a bridge there should be a warning if there is an enemy nearby.***

At this point his programming description is still far from including an idea of a concrete implementation. A major obstacle for J throughout this episode will be to move from talking about his game within a narrative framework to talking about it in terms of detailed design, and later in terms of implementation.

J's first idea was quite elaborate and clearly based on how he wants the existing narrative of the game to be made more interesting. He is concerned that the game is not really good as it is (turn 20) and suggests that a warning behavior should be added to the cheese so that when the player character eats the cheese, the player gets this behavior as well (turn 21). His suggestion for how to add the warning behavior is phrased in terms of robots and programming, however it also refers to the narrative elements of the game, such as a bridge, an enemy, and a warning. To turn this into an implementation a distinction between the narrative and the programming elements of the game are required. Fragments of this can be seen here as he talks about the original magic cheese in terms of sub behaviors: the general transfer behavior, and a behavior that gets transferred, in this case a warning. Thus, he has started to describe the behavior of magic cheese, as a general mechanism for transferring any kind of behavior to other objects, and not only as a instance specific behavior which provides the player character with vertical movement. The cheese is not only something that you acquire the ability to move vertically from. It is something that can be rebuilt to give the player character any behaviour you find appropriate.

Thinking about different implementations

The transformation of the warning behavior (described loosely as in the previous section) into a more detailed plan for implementation turned out to be too complicated for J at this point. Hesitant comments from R contributed to J giving up the 'warning' behavior, as initially talked about. However, he continued to want to use the transfer behavior and got interested in the "move diagonally" behavior.

28. ***J - [Takes out the move diagonally behavior for making things move diagonally and flips it over]***
29. *R - Then one can move …what would happen if you had one of those?*
30. ***J - Well, you would be able to move in all directions***
31. *R - That's pretty good, want to try?*
32. ***J – Yes***
33. *R - see if it works?*

After J played with the 'move diagonally' (turn 28) behavior R asks him (turn 29) to think about how the new compound behavior would turn out if the 'move diagonally' behavior was combined with the original cheese behavior. J replied that you would be able to move in all directions (turn 31), i.e., previously you could move horizontally and vertically after hitting the cheese, but if the 'move diagonally' behavior is added the player would be able to move in all directions. J's prediction regarding the new compound behavior includes a combination of the original magic cheese with a behavior for controlling objects with four different keys.

To predict how a combination of behaviors will be manifested in the game is of central importance in the process of planning the implementation of game. Another aspect of this was discussed above when J constructed an explanation for how a behavior in a game could have been realized in code. These are two important aspects in processes of design of software and interactive artifacts since they involve explanations of program behaviors in narrative terms, as well as predictions of how the behavior will finally run.

Collaborative implementation of the behavior

The first step in implementing the behaviour is to identify the correct program components.

1. R – So how do you do that? You need to suck out that one, right?
2. J – Yes, that one [points at 'move diagonally' behavior]
3. R – Yes, then we have to –
4. J – =[J starts pointing with Dusty]
5. R – the whole one
6. J – Here, [sucks out 'move diagonally' behavior]

Here, J implements the behavior he predicted in the previous section. Interestingly, even though this mostly involves silent actions from J, a lot of communication happens between J and R through the elements of the programming environment. In these turns (starting in 2) J seeks confirmation from R by starting out programming actions such as pointing to take something off another picture (turn 4), which is confirmed by R with a slight correction to make sure that J gets "the whole" picture (5). In turn 6, J completes the action he initiated earlier by taking out the correct behavior and confirming by saying 'here'.

Even though R in a sense asks questions regarding programming issues, he seldom verbalizes these in natural language. Programming requires focused attention, which makes it difficult to explain one's own programming actions and simultaneously completing them. Instead J communicates regarding programming through the interface of the programming language. The programming of this new behavior requires a number of detailed steps which J and the researcher carry out collaboratively (see Figure 5).

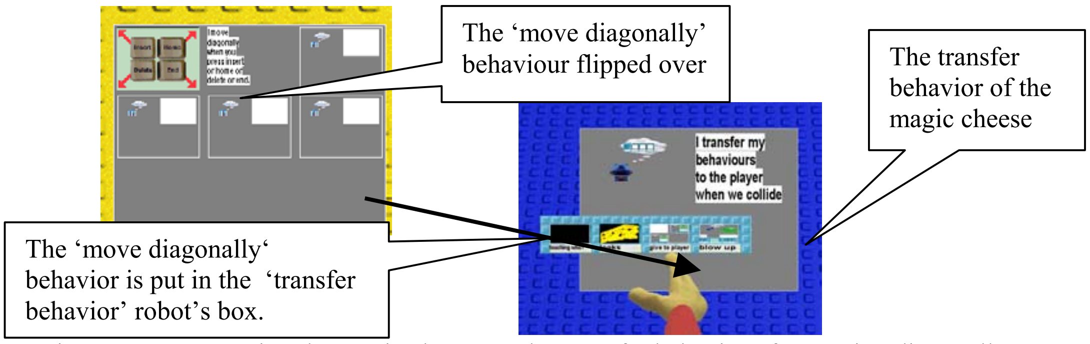
The 'move diagonally' behaviour flipped over
The transfer behavior of the magic cheese
move diagonally when you press insert or home or delete or end
I transfer my behaviours to the player when we collide
give to player
blow up
The 'move diagonally' behavior is put in the 'transfer behavior' robot's box.

Figure 5: Programming the magic cheese to also transfer behaviour for moving diagonally

The first necessary step was to add the 'move diagonally' behavior to the hole in the robot's box where the behavior that the player character gets, is kept. This is achieved in a number of exchanges (not included here) where the guidance from R is crucial for J in order to complete the necessary steps. However, J does not blindly follow instructions from R. Rather, he knows what needs to be done even though he needs support to get the details right. This requires intensive collaboration and coordination between the J and R. Most programming actions are initiated by questions or comments from R however J often carries out more than is explicitly expressed in the instruction from R. To follow this kind of instructions is not just simply to carrying out what is expressed. It requires a recontextualization of verbal descriptions into the current situation of use, which has been shown to be a most qualified task (Suchman, 1987). Even though J needs quite extensive support to complete these actions, he has an elaborate idea of the environment, which allows him to complete them efficiently. A novice without any knowledge of Playground or TT would not be able to follow this kind of instructions.

Putting the game back together

The implementation of the new behavior was the first step in completing the change so that it could be fully integrated in the game. The new magic cheese now had to be put back in its original scene to make the change complete. However, in order for the changed scene to be incorporated into the game J had to put it back into the scene notebook used by the scene changer. This requires the programmer to simultaneously keep track of several program elements and how they are connected. This is cognitively demanding, especially for an eight-year-old child with a limited understanding of the system as a whole. In the following excerpt R guides J to complete the required changes.

25. R – And then we have to put it back in the notebook. So it becomes the one we are using.
26. J - [gets scene notebook from the back of the game and flips to page with cheese scene] here?
27. R – and then you have to change that [point to cheese scene]
28. J – mmm [sucks out old cheese scene and replaces the changed one]
29. R – because they are different

Similar to the previous dialogue R initiates programming actions that J completes. In turn 25, R states that the changed scene must be put back in to the scene notebook. J confirms this (turn 26) and also shows that he knows the next step since he flips to the page in the notebook where the scene with the original cheese is kept.

However, he seeks support from R by asking him to confirm that this is where he should put the changed scene. R answers by saying that he must change the scene. This is probably what J expected since he confirms silently and removes the old scene and also puts the new one in its place. R confirms that J is doing the right thing this by saying that this is done because the two scenes are different (turn 29), which is a correct motivation.

J and R continue with similar sequences of initiation and confirmation, until the game is put back together. When the game is reassembled J tries out the new cheese scene in different ways to make sure that it works the way he had intended, which it does. The new compound behaviour of the magic cheese and the player character behaves like he predicted.

Most of the verbal actions in the sequences discussed above are in the form of questions and instructions from R which J responds to. However, without an elaborated understanding of this particular game and how the programming environment works it would be very difficult to follow these instructions. It is not a goal in itself that children of this age should develop a complete understanding of the programming environment from a computer science point of view. But, it is still of great importance that learning environments with the intention that children should engage in programming like activities actually support a meaningful use of the programming language, preferably in a way that can be used in other contexts.

Part 3: Starting to change the scene structure of the game

After the intensive work with the transfer mechanism J decided that the next thing to do was to add a new portal to one of the scenes. We talked about a fun and interesting place for a new portal and J suggested the start scene. It would take the player character to the scene with a hamburger and provide an alternative route to get past the poisonous frog. J quite independently completed the necessary changes to the start scene and to the scene notebook to make this happen. As he started playing the game he realized several consequences related to the design of the game. One was that the game was virtually impossible to complete if the first thing you did was to choose to go through the new shortcut. It would make it impossible to reach the cheese since you could only move horizontally. To complete the game you first had to take the original route and then go back to the shortcut.

These considerations also involve important aspects from a design and programming point of view. Predicting consequences when introducing a new behavior is often difficult even for experienced programmers. Here, this involves a distinction between the narrative and the programming dimensions of a game as a major obstacle that the children must overcome. Clearly, J's understanding of the game mechanisms is constructed through a combination of imagination, programming, and playing. He cannot predict all the consequences of his design choices in advance. Rather, he made changes to the game and evaluated their consequences by it. This helped him to understand the consequences of his design choice, which was reflected in how he named his game. He named it "The fool", providing a clue that in order to complete his game "you have to make the right choice".

Adding a new scene to the game

During the second session J was given a task concerning the construction and incorporation of a new scene into his game. This involved a range of different considerations regarding design and programming. It not only involved adding a scene to the scene notebook controlled by the scene changer, but to fully complete the change other considerations such as how to connect the new scene to the other scenes were also required. Throughout the following three excerpts, J's talk about the new scene he is building gradually became more specific with respect to the design of the game. Initially his ideas about the scene are somewhat fuzzy, but through interaction with the game and collaboration with R, J articulates his ideas in terms of the narrative, as well as implementation aspects of the game.

Design talk.

The discussion initially concerned how the new scene should be connected with the other scenes and J stressed that the new scene should fit in with the rest of the scenes. His way of talking about the structure of the game indicates that an understanding of which does not require simultaneous interaction with the game elements.

1. *Researcher - Do you remember the comment for your game from the last time? "You have to make the right choice"*
2. ***J - Yes, if you go that way, then you are out.***
3. ***J - I was thinking of adding the scene so that if you go, there (from the hamburger scene to the new scene)***
4. *R – Okay, shall we draw it here*
5. ***J – then it's going to be this scene [points to the top of the new scene which he had started at the previous session]***
6. *R - So you are supposed to come from up above*
7. ***J - Yes from there [points at the top of the new scene]***

Here J is making several considerations to ensure that the game is consistently designed with respect to the original game. In turn 2 he talks about the change he made previously and how that required the player to make strategically correct choices in order to complete the game. This leads the discussion to move from a strict focus on design at the narrative level to also involve considerations at the level of implementation.

Structured design talk.

The talk regarding the integration of the new scene into the game and about how it should fit in with the rest of the game evolved into a quite long discussion. This is a typical example of how work in the Playground environment often lead to structured talk about topics that were not immediately related to programming issues, but still important to consider when designing computer games or other artifacts. Previously, most of the work has been focused on the programming environment and much of the interaction between J and R were conducted through the programming tools with only minor conversational elaboration from J. However, in the following excerpts J gradually becomes verbally more articulate and at the same time less focused on programming details and instead concerned with game design. It is interesting to note that J reasons and elaborates around the consequences of different design choices without having to practically construct them himself, as was the case during the previous stages. This indicates that he has developed an understanding of the game that is not only constructed *in situ* as it is played. He can now also refer to the game from different perspectives, as something that is played, well as something with an internal structure that can be changed which in turn has consequences for how the game is played. After thinking about the consequences of his decision (that the player character are supposed to come from up above) for the whole game he hesitates.

8. J – ***But let's change it like this instead… no then one cannot go up, so***
9. R – Why not?
10. J – ***Because you cannot get the cheese***
11. R - You can add it!
12. J - ***Yes, we can add a cheese there***
13. R - And then you are able to go there

R suggests to him that he can add the cheese on that scene in order make it possible to acquire the vertical movement (turn 11). However, he is not satisfied with this which he elaborates on in next section. The movement from the hamburger scene to this scene would give the player the feeling of getting there through a vertical movement which would be impossible without that behavior and would be inconsistent with how the player character got there. In the following J continues to think about what should happen on the new scene and what tools the player character needs to bring from the other scenes to complete the scene and continue the game.

Detailed design considerations.

J further evaluates his ideas in light of how they would be programmed. There are clear reflections of the underlying mechanisms of the game in the ideas which J expresses.

14. R - I was thinking that when you continue from here where do you get to then?
15. J - To the last scene
16. R - To the treasure
17. J - Yes, or you should be stuck
18. R - You should get stuck?
19. J - If you take the wrong way yes. Then we have to make it with the hamburger. When we get to the hamburger then, then we put a portal so one can go down [to the new scene], then you come here [points at the top of the new scene]
20. J - The new portal has to be down there since we come from the top

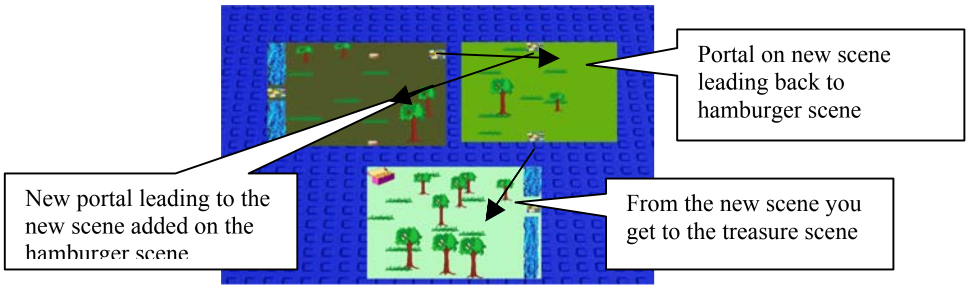
Portal on new scene leading back to hamburger scene
New portal leading to the new scene added on the hamburger scene
From the new scene you get to the treasure scene

Figure 6: Incorporating the new scene into the game

He decides that this should be the last scene before reaching the treasure (turn 14). This leads him to the idea that a suitable place for the new scene is after the hamburger scene, so that it is possible for the player to get a behavior for shooting before getting there (see Figure 6). J does not randomly add objects to his game he has specific intentions with the different elements that he wants to add and the sequence of those elements in the game. By including a dead end (turn 19), he tries to include elements of strategy to the game since he wants to make sure that the new elements give the player a realistic impression of the game as a whole (turn 20).

This long exchange between R and J is interesting in terms of learning design and programming because there are clear indications that the structured nature of the adventure game together with the programming environment supports J to think about his game in a way that forces him to be well-structured around the elements he wants to include: e.g. how new elements connect with the rest of the game, and how he can use program language elements to implement his ideas. If anything of the underlying structure that drives the game is left out, the game will not be played as intended. Furthermore, if new elements are included without a well considered purpose with respect to the overall design of the game it will not be interesting for another player of the game. The fact that the task involves a balance between elements of design as well as programming augments a structured way of thinking around the game.

Summary and Conclusions

J made a series of significant changes to the original adventure game. These were not trivial or cosmetic. He combined, altered, and constructed new behaviours, and integrated these back into the complete program. To J the adventure game is not like shrink-wrapped software that one treats as a black box. Instead it is software that he can pull apart, understand its internal mechanisms and design, and mould to his liking. He can learn about complex mechanisms, discover the internal structure of a behaviourally rich artefact, and be creative. However, the most important aspect of J's work is not his impressive handling the environment. The interesting aspects in terms of learning was how he were able to describe the general properties of programming elements and that he could move between design and programming dimension of his game. Two issues were particularly important in facilitating these activities. First, the intense collaboration with R allowed J to engage in programming and design activities that would be impossible otherwise. Second, the concrete interface metaphor provided means for J to communicate regarding programming in non-verbal ways.

The human support and the interface metaphor

A significant amount of adult support is required in order for a child to program in the Playground environment. The kind of software that was investigated in this paper would perhaps be of limited use to children working individually. A significant problem for J, and for most children we have worked with, is to keep track of the details required to get all the program elements in place. The details of changing or constructing new behaviors most often require detailed guidance from an adult, which is in line with a lot of the research on Logo (Clements & Meredith, 1992). However, adequate support by an adult makes it possible for children to get in touch with, and manipulate programming concepts that otherwise would be out of their reach. On behalf of the adult, this requires rather detailed skills of the programming environment and the game currently worked with,. One could object that there are serious limitations in an environment like this, since the required adult- child ratio is unrealistic in many learning situations. However, most Logo environments lacked an interface metaphor that let children interact with program language elements in the way seen here.

The interface of TT and Playground provide a very concrete metaphor for manipulating program elements. The virtual artifacts, with properties resembling everyday objects (such as play robots, boxes, and, holes) serve as mediators in children's construction of verbal explanations of program behaviors, and when thinking about how to implement behaviors. These elements make it possible to refer to programming elements without having to master the details of a traditional programming language, with the abstract terminology often involved. Instead of having to speak about variables, objects, or values, such elements can be referred to through pointing, and through grabbing things. Even though it is not expected that children of this age should learn how to master a program language of this complexity, it is still possible to make it feasible for children to communicate regarding elements in the language, given an interface design that does not require a full conceptual understanding of the underlying properties of the language.

However, a dilemma arises through such localized ways of referring to programming elements. The concrete programming metaphor which allows programming "without words" also limits the development of a more general vocabulary which is more easily transferred to other domains A concrete interface metaphor lets children construct advanced computer programs by using a simple terminology such as “there”, “these”, or “inside. This would be virtually impossible in more abstract or general purpose programming environments. On the other hand these concrete, often non-verbal, ways of “talking” about programming limits the possibility of developing an understanding of concepts that can be used in other domains. A knowledge, such as the one developed here, which involves an indexical style of talk, is specifically tied to the environment and to the artifacts used, which makes it difficult to use in other domains. Theoretical programming concepts such as variables, loops, and iterations are more appropriate to transfer to new contexts and situations because this is one of the purposes they were developed for. However, they have been shown to be complicated for children to grasp. One way of overcoming this dilemma of concrete versus abstract programming can be found in the human support that children are given. The role of the tutors must be to encourage the children to think of general aspects of their programming activities, for instance by suggesting how a game programming activities can be useful in other situations. The processes involved in these issues need to be further explored.

References

Clements, D. H., & Meredith, J. (1992). *Research on Logo: Effects and Efficacy*. Logo Foundation [2002-07-03].

diSessa, A. (2000). *Changing Minds: Computers, Learning, and Literacy*.: MIT Press.

Kafai, Y. B. (1995). *Minds in Play: Computer Game Design As a Context for Children's Learning*.: Lawrence Erlbaum Assoc.

Palumbi, D. B. (1990). Programming Language/Problem-Solving Research: A Review of Relevant Issues. *Review of Educational Research, 60*(1), 65-89.

Rader, C., Brand, C., & Lewis, C. (1997). *Degrees of Comprehension: Children's Understanding of a Visual Programming Environment.* Paper presented at the CHI, Atlanta.

Roschelle, J. (1996). Convergent Conceptual Change. In T. Koschmann (Ed.), *CSCL: Theory and Practice of an Emerging Paradigm*.: Lawrence Erlbaum Associates.

Suchman, L. (1987). *Plans and Situated Actions*.: Cambridge University Press.