Conversational Programming in Action

Alexander Repenning
AgentSheets Inc.
Boulder 80301, Colorado, USA
alexander@agentsheets.com

***Abstract* – Accelerated by the Do-It-Yourself mindset of the Web 2.0 culture, end-user programming, which is programming by end users with limited, if any, formal programming background, is growing rapidly. Especially in educational settings, children are exposed to computational thinking by making games, building scientific simulations and creating stories. Early educational programming languages such as Logo have made programming substantially more accessible to end users. More recent approaches include visual programming with drag-and-drop style of programming making it nearly impossible to compose syntactically incorrect programs. However, as the syntactic challenges of end-user programming are gradually fading into the past, the new frontier of semantic programming support emerges. This demonstration introduces Future Trace, a system to make programming more conversational. A conversational programming agent runs programs one step into the future in order to visualize discrepancies between the programs users intended to write and the actual programs.**

Keywords – Game design; computational thinking; debugging; end-user programming; visual programming.

I. Towards Conversational Programming

In programming the interaction between the programmer and the programming environment is typically asymmetrical and often limited to syntactic feedback, which is limited in nature to programs that are malformed. Miss one semicolon in a C program and the program may no longer work at all. A programmer may spend considerable amount of effort to write a program before the programming environment provides meaningful feedback.

One idea to simplify programming would be to make the communication process between the programmer and the programming environment more symmetrical with the goal to aid debugging. But just how can one conceptualize debugging? Pea [1] describes the process of debugging as:

> *systematic efforts to eliminate discrepancies between the intended outcomes of a program and those brought through the current version of the program.*

A number of programming approaches including programming by example and natural programming try to systematically reduce these discrepancies by having programmers demonstrate actions on concrete examples or by providing programming languages that more closely resemble the way users with no programming background tend to think about certain problems. A different approach is the notion of *conversational programming* employing computational agents to provide real-time semantic feedback to a programmer so that the programmer can indentify discrepancies between the *intended program* and the *actual program*.

Conversational programming could be conceptualized as a simple form of pair programming substituting a human partner with a computational agent called the *Conversational Programming Agent* (CPA). Figure 1 describes a conversational programming architecture.

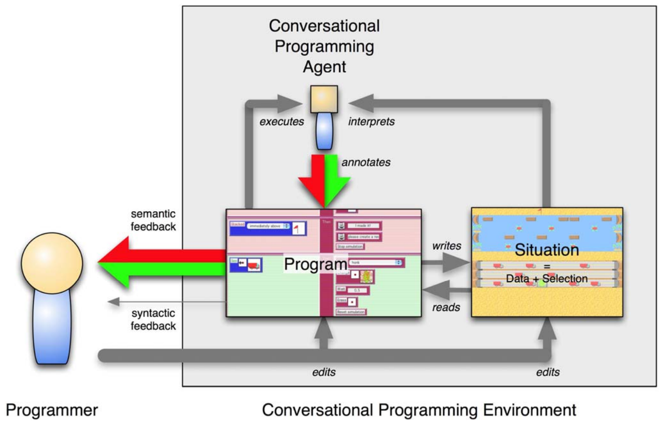


Figure 1. Conversational Programming: The programmer edits the program and edits the game world. A conversational programming agent executes the program, interprets the situation and annotates the program semantically.

The notion of a conversation suggest the need for a:

- ***programming partner/agent*** capable to serve as that other pair of eyes.
- ***more symmetrical and semantic interaction*** between the programmer and the programming environment.
- ***shared context with a defined focus*** corresponding to a conversation topic. For instance, a programmer should be able to select an object in a game world to make the conversation relevant to this object and its state.

The goal of Conversational Programming is to reduce the discrepancies between intended program and actual program by using notions of conversations to make the interaction between the programmer and the programming environment more symmetrical, more timely, and more meaningful.

II. Prebugging: Proactive Debugging

The specific implementation of conversational programming presented here is called *Future Trace*, which is integrated into the AgentSheets [2] game and science simulation end-user programming tool. Visual AgenTalk is the drag and drop, rule-based programming language of AgentSheets with a long history in educational applications going back to 1994. AgentSheets is a popular game design tool used in public schools especially at the middle school level.

Novices, such as middle school students building their first game with no programming background, as well as more advanced programmers, such as computer science undergraduate students, often exhibit difficulties when trying to understand complex rules. For instance, confusion resulting from the order of instructions is surprisingly common and is not limited to beginning programmers [3]. Common questions include: why does this rule fire? Why does that rule NOT fire? Why is this condition or rule not even being tested? What is the order in which conditions and rules are tested? Why is the rule and condition order of fundamental importance?

Future Trace could be considered a *prebugging* tool [3] providing answers to the questions above even before the program is completely written or executed. In Future Trace the Conversational Programming Agent (CPA) will proactively execute parts of the program created by the programmer and annotate it discretely in order to support the end-user in recognizing potential differences between the intended program and the actual program. The CPA focuses on the agent selected by the user in the game world and visualizes the outcome of running the program of the selected agent one step into the future. For instance, if the programmer had previously selected the only frog in the worksheet (Figure 2, left) then conversational programming annotations suggest that the frog is about to be crushed by the truck.

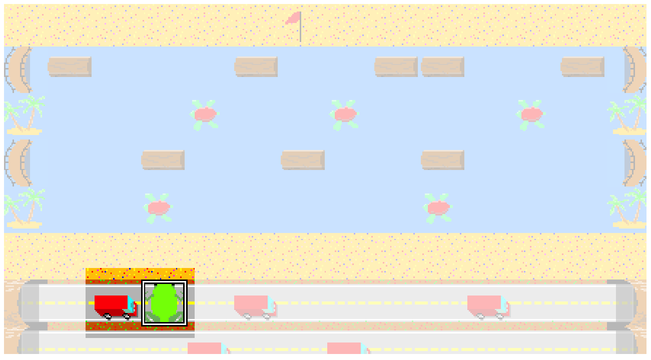
The presentation will share some of the evaluation data from middle school teachers, middle school students and computer science undergraduate students. Further, it will present test cases in which conversational programming provides significant programming and debugging advantages which are not typically found in traditional drag and drop end-user programming environments.

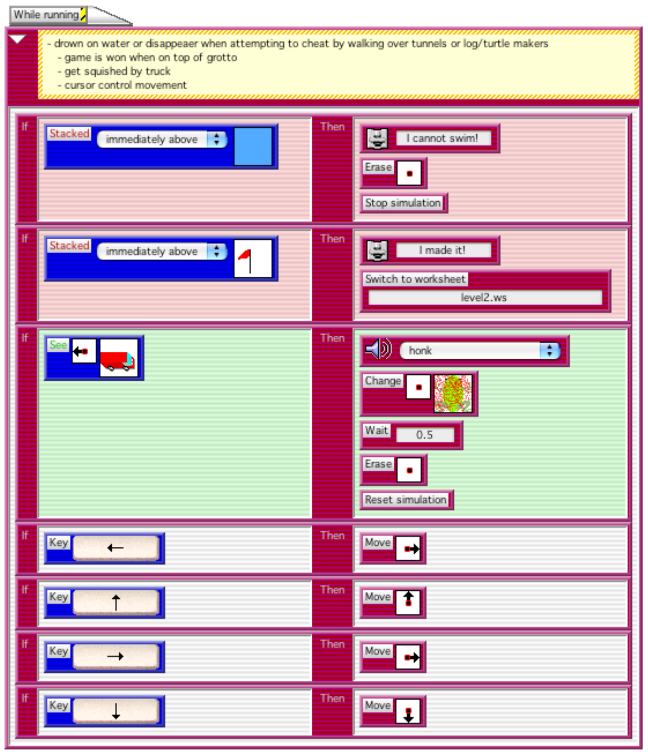
III. Acknowledgements

This material is based in part upon work supported by the National Science Foundation under Grants DLR-0833612 and IIP-0848962. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

IV. References

1. Pea, R. D. Chameleon in the Classroom: Developing Roles for Computers, Logo Programming and Problem Solving. In Proceedings of the American Educational Research Association Symposium (Montreal, Canada, April 1983) (Montreal, Canada, 1983).
2. Repenning, A. and Ambach, J. Tactile Programming: A Unified Manipulation Paradigm Supporting Program Comprehension, Composition and Sharing. Computer Society, City, 1996.
3. Telles, M. and Hsieh, Y. The Science of Debugging. Coriolis Group Books, Scottsdale AZ, USA, Scottsdale, 2001.





- drown on water or disappeaer when attempting to cheat by walking over tunnels or log/turtle makers
  - game is won when on top of grotto
  - get squished by truck
  - cursor control movement

Figure 2. The truck will crush the frog selected in the worksheet (left). Rules 1 and 2 of the Frog behavior (right) are tested but contain at least one condition keeping them from firing. All conditions of rule 3 are true. Rules annotations (background): green=would fire, red=would not fire, and gray=not tested; Conditions annotations (text label): green=is true, red=is false, black=not tested.