LEGOsheets:
A Rule-Based Programming, Simulation and Manipulation Environment for the LEGO Programmable Brick

Jim Gindling, Andri Ioannidou, Jennifer Loh, Olav Lokkebo, Alexander Repenning

Department of Computer Science
Center for LifeLong Learning and Design
University of Colorado, Boulder CO 80309-0430
(303) 492-1349, ralex@cs.colorado.edu
Fax: (303) 492-2844
http://www.cs.colorado.edu/~ralex/public_html/Home.html

Abstract

The LEGO[1] Programmable Brick gives children the ability to create physical artifacts, such as vehicles and robots, and program them with interesting behaviors. However, programming is difficult to learn, even for adults. Children often lose interest in further exploration of programming through adult learning mechanisms. Environments that support a gradual transition from manual control of the physical artifact to complete programming substantially simplify the process of programming. The combination of LEGOsheets and the Programmable Brick is an educational environment that provides a gentle, enticing introduction to programming and the design of mechanical artifacts. This paper introduces LEGOsheets, a rule-based programming environment that allows children to simulate and manipulate the LEGO Programmable Brick.

1. Introduction

The LEGO Programmable Brick, presently being developed at the MIT Media Lab, is a small but complete computer, that can be used in conjunction with sensors and effectors to program the behavior of mechanical artifacts. However, it is difficult to program and debug using traditional programming environments, such as the current programming interface. To address this problem, LEGOsheets was created. LEGOsheets is an educational environment implemented in Agentsheets, a grid-based tool for creating visual programming languages [5]. Programming can serve as a vehicle to create learning opportunities in the constructionist sense [3].

Making use of a grid-based spatial construction paradigm [4], LEGOsheets is a combined visual *programming, manipulation, and simulation* environment. It provides a gentle and enticing introduction to programming and the design of mechanical artifacts. LEGOsheets is intended to be used by children ages 8 and up. Initially, children use LEGOsheets simply as a direct manipulation [6] mechanism to control physical artifacts equipped with electrical sensors and effectors. During this stage, LEGOsheets can be perceived as a fancy remote control. Later, children use a rule-based approach [2] to program the behavior of individual effectors, such as motors and lights, while still manually controlling others. In the final stage, all the effectors are programmed. The program gets downloaded into the Programmable Brick, the link between the programming environment and the Brick gets severed, and the artifact, controlled by the Brick, is let loose. We believe this gradual transition from manual control to complete programming serves as an ideal method for children to learn how to program. This paper first introduces the LEGO Programmable Brick, then gives a brief description of a LEGO vehicle built and programmed using LEGOsheets, and finally walks through a scenario with children using LEGOsheets.

[1] LEGO is a registered trademark of the LEGO Corporation.

2. LEGO Programmable Brick

The LEGO Programmable Brick is a small computer with sensor ports and effector ports (Figure 1). It is currently being developed at the MIT Media Lab and is not yet commercially available. To program the Brick, the MIT Media Lab created a system called Brick Logo (Figure 2).

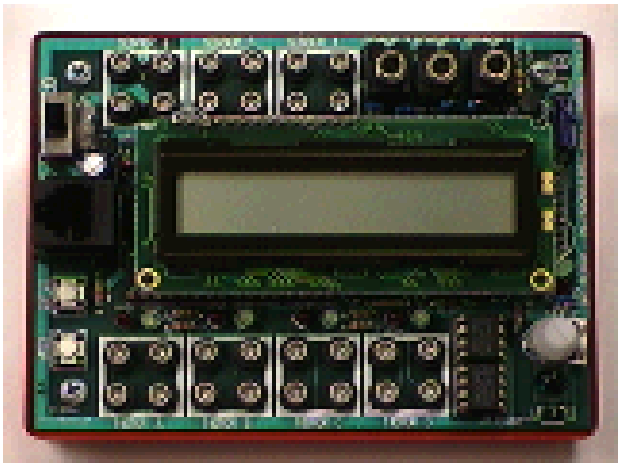

Figure 1: The LEGO Programmable Brick.

Through the sensor ports, the Brick can receive signals from a variety of sensors, such as touch sensors, reflectance sensors, and angle sensors. Through the effector ports, it can control motors, beepers, and lights. The Programmable Brick also has a few built-in sensors and effectors, including an infrared sensor and a beeper.

Over the years LEGO has presented a variety of construction kits that can be grouped into three generations. The first generation construction kits (the traditional LEGO bricks) allowed children to build static objects such as buildings. The second generation (the Technic[2] bricks) enabled children to build more sophisticated artifacts using motors and pneumatic devices. With the introduction of the Programmable Brick, we enter the third generation, which allows children to build behaving machines [7]. The size of the Brick allows it to be incorporated in LEGO constructions and therefore create a wide variety of interesting machines and creatures. One can build data acquisition devices such as a weather station or a device that can be placed by a door to count how many people enter a room. Furthermore, one can build a vehicle, or LEGO creature, that explores its environment by programming it to find the place with the most light or the highest temperature.

3. Building the LEGOpet

In this section, we provide a scenario to show how LEGOsheets is used to program the Brick. The scenario is based on our experiences during user testing. Throughout the design and development of LEGOsheets we have met on a biweekly basis with five middle school students, grades 6 through 8. During the first stages of user testing the children explored existing software such as LEGO Dacta[3] and Brick Logo. The feedback that they gave us during this stage made them part of our collaborative design process, which resulted in the prototypes we developed.

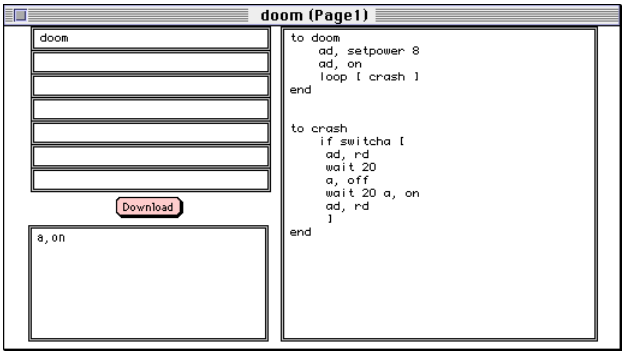

Figure 2: The Brick Logo Environment for Programming the Brick.

Our experience when programming the Brick with the existing environment, Brick Logo (Figure 2) during user testing, indicated the need to facilitate a more gradual transition from manual control to programmed control. The scenario in section 3.2 illustrates this gradual transition.

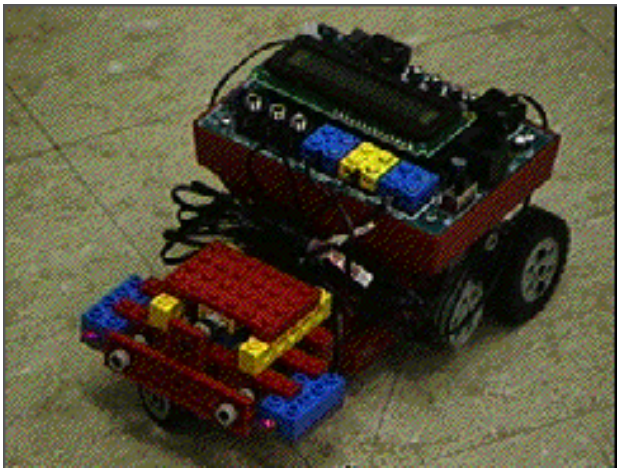

Figure 3: The LEGOpet.

---

2 Technic is a registered trademark of the LEGO Corporation.

3 LEGO Dacta is a registered trademark of the LEGO Corporation.

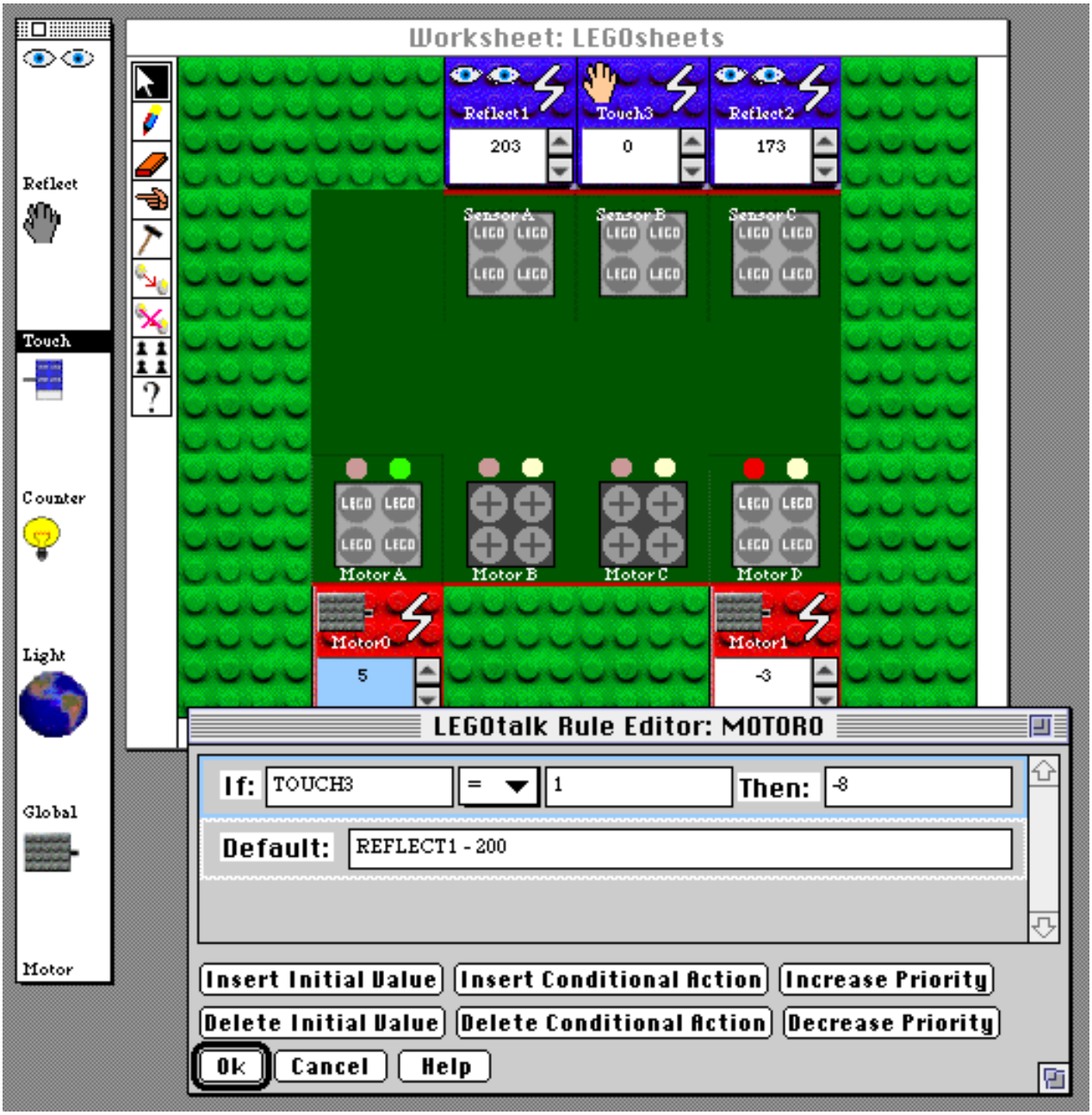

Figure 4: The Gallery, Simulation Environment, and Rule Editor of LEGOsheets.

The LEGOpet (Figure 3) is one of the LEGO creatures the authors created and programmed using LEGOsheets. We created the LEGOpet by building a vehicle with a touch sensor, two reflectance sensors, and two motors, and then assigning behavior to it. We programmed the LEGOpet in such a way that it "likes" dark objects and runs toward them, but is "afraid" of light-colored objects and backs up whenever it approaches one. The idea of vehicles exhibiting behavior and feelings is explored in Braitenberg's *Vehicles: Experiments in Synthetic Psychology* [1].

3.1. LEGOsheets Components

When the user launches LEGOsheets, the *gallery* and *simulation environment* appear on the screen (Figure 4). The *gallery* contains icons representing sensors and effectors that can be connected to the Brick, as well as other components that can be used for programming. The *simulation environment* contains the *virtual Brick*, a realistic representation of the physical Brick able to completely simulate the Brick. Users can attach components of their choice from the *gallery* to the *virtual Brick* in the *simulation environment* and start playing with

LEGOsheets. In Figure 4, a touch sensor is connected to Sensor Port B and two reflectance sensors are connected to Sensor Ports A and C, and two motors are connected to Effector Ports A and D.

In order to assign behavior to the different components, users define rules. The rules in LEGOsheets are attached to each individual effector, and consist of three major components: an *initial value*, zero or more *conditional actions*, and a *default action*. The *initial value* is optional and contains the value assigned to the effector at the beginning of program execution. The *conditional actions* are if-then constructs. If the condition holds, the value of the expression specified in the "then" part gets assigned to the effector. Only one of the *conditional actions* is executed each time the rule is checked. If none of the conditions holds, the value specified in the *default action* is used.

3.2. Scenario

The following is a scenario based on our experiences during user testing, in which two children, Christine and Jim, build the LEGOpet. Christine is older. She has used LEGOsheets a few times before and is relatively familiar with the system.

The children start out with a vehicle that has two motors, each used to drive one wheel. It also has two reflectance light sensors, which test the reflectance of objects, and a touch sensor, which can detect when the vehicle is in contact with another object.

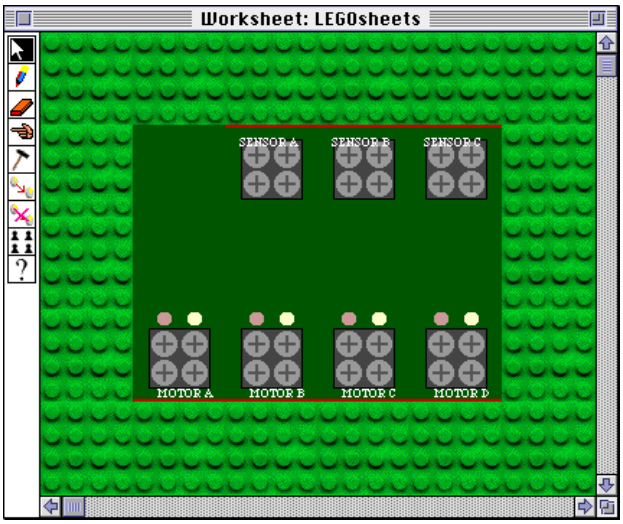

Figure 5: The Initial Simulation Environment.

When Christine and Jim start LEGOsheets, the simulation environment contains only the virtual Brick (Figure 5). The first thing they do is add two motors to the simulation environment (Figure 6). As the motors are inserted they make a snapping sound in the same manner LEGOs do when they are put together. Throughout user testing, we have learned that children appreciate colorful icons and sounds that imitate the world around them. Thus, we believe the use of color, animation, and sound to be important forms of feedback to the user.

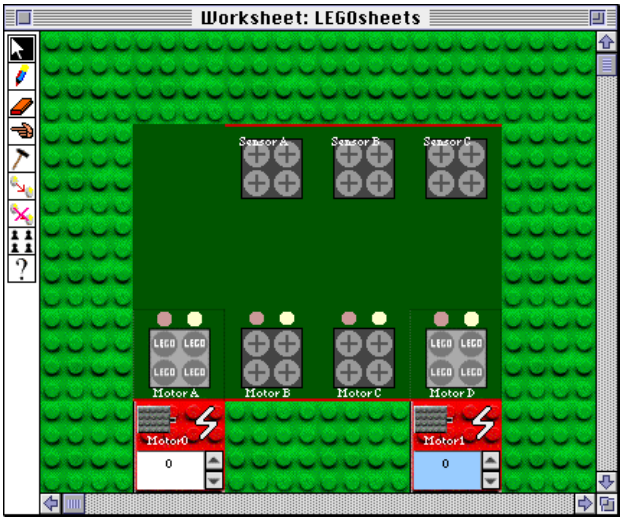

Figure 6: Simulation Environment with 2 Motors Attached to the Virtual Brick.

3.2.1. Direct Manipulation of the Physical Artifact

The LEGOsheets environment allows programming of the *virtual Brick* without the physical Brick. Furthermore, it allows the user to directly manipulate the physical Brick through the *virtual Brick*. Virtual sensors can be simulated by the user or receive real values from the physical Brick connected via a cable to the virtual Brick. The same holds true for effectors.

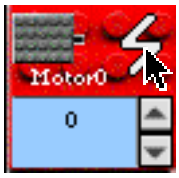

Figure 7: Virtual Motor Before Being Wired to Physical Motor.

After playing around with the values of the virtual motors in the simulation environment, the children want to see the real motors spin. Christine checks to make sure the Brick is connected via the serial port to the computer. They click on the part of the virtual motor that looks like a lighting bolt (Figure 7). The lightning bolt makes an electrifying sound and changes color (Figure 8). Once again, the change of color and the sound produced as an

indication of wiring the motor proved to be useful feedback on the change of state of the environment.

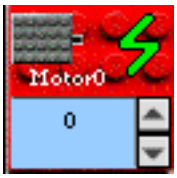

Figure 8: Virtual Motor After Being Wired to Physical Motor.

Then Jim wires the other virtual motor to its corresponding physical motor. Virtual sensors and effectors can be wired. For sensors, this means that they get their value from the corresponding sensor on the physical Brick. Effectors update their corresponding physical effector. Now that Jim has wired the virtual motors to the physical motors, he clicks on the increase-value arrow for one of the virtual motors (Figure 9). The corresponding physical motor turns on at a very slow speed. He notices the value displayed on the virtual motor is now 1. He clicks the increase-value arrow for the same motor until the value displayed is 5, and notices that the physical motor increases speed each time he clicks. He then clicks the decrease-value arrow for the virtual motor until the value displayed is -2. He notices that when the value is 0, the motor stops, and when it is negative, the motor reverses direction.

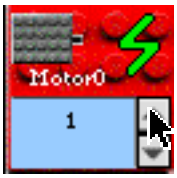

Figure 9: Virtual Motor After Increase-Value Arrow is Clicked.

As the children change the motor values, the vehicle moves around on the table and hits a book that lies in its path. They try to make the vehicle climb over the book and discover that when the motors are set at speed 5, the vehicle is able to slowly climb the book. Christine then increases the angle of the book and, after experimenting with different values, they find it now requires the motors to be at speed 7 to climb the book.

While directly manipulating the physical Brick, children can experiment with effector values and use the knowledge they acquire later, when specifying rules.

3.2.2. Combining Direct Manipulation and Programming

At any time, individual effectors can be switched between user-controlled and programmed mode. This supports the gradual programming of behavior.

At this point Christine suggests they write a program. She explains to Jim that motors have rules attached, and writing programs requires editing these rules. She continues by stating that to edit the rule for a specific motor, he must first double-click on the motor so its *rule editor* will open. Jim double clicks on one of the motors, and its associated *rule editor* opens (Figure 10).

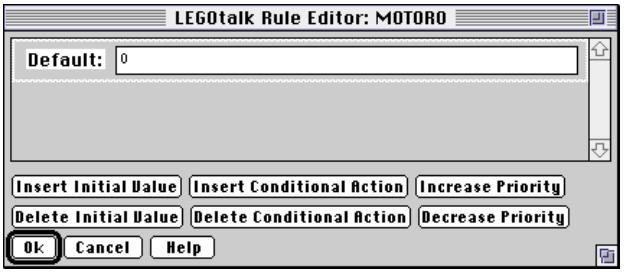

Figure 10: Default Rule Editor for a Motor.

Christine informs Jim that this rule currently specifies that the motor is always at speed 0, which Jim knows will result in the motor being turned off. Realizing that this is not interesting behavior, Jim changes the 0 to a 5. He then clicks the OK button, and the rule editor closes. He opens the rule editor for the other motor and does the same. Then Christine starts the execution of the rules by selecting the *Run Rules* menu item. Once the vehicle, which faces the wall, runs into it, Jim picks it up and turns it around. Christine, however, suggests that they stop the rules and reverse the motors manually, so that the vehicle will back up from the wall.

After a while, Jim decides that he wants to play with the sensors. To the *simulation environment* (Figure 11) he adds two reflectance sensors, which correspond to the physical sensors attached to the Brick.

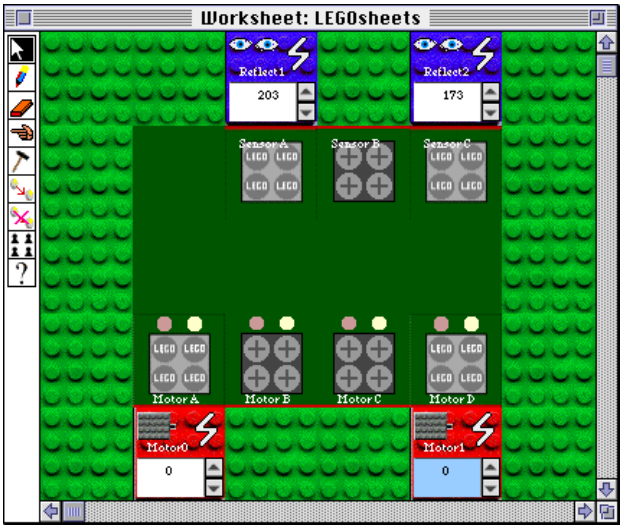

Figure 11: Simulation Environment with 2 Motors and 2 Reflectance Sensors Attached to the Virtual Brick.

Jim notices the values displayed on the virtual sensors are 0. Because the reflectance sensors also have lightning bolts, he wires the virtual sensors to the physical sensors by clicking on the lightning bolt of each. Now the values displayed on the virtual sensors are around 200, but change between about 198 and 202 frequently. Jim puts a piece of white paper in front of one of the physical sensors, and the value of the corresponding virtual sensor changes to about 190. He removes the piece of paper and the value jumps up to around 200 again. He repeatedly places the paper in front of the sensor and removes it, and watches the value change. This reminds Christine of the plot mode and she applies the "hand" tool , from the toolbar on the left of the *simulation environment*, to change the value representation of the reflectance sensor (Figure 12).

Figure 12: Virtual Reflectance Sensor Before Being Changed to Plot Mode.

The plot mode in LEGOsheets is an alternative way to represent the value of cells. Whereas in the numerical mode the value of the cell is displayed as a number, in the plot mode the value of the cell is displayed as a graph plotted over time.

Figure 13: Virtual Reflectance Sensor in Plot Mode.

Jim takes the paper and watches how the line changes shape as he moves the paper in front of the sensor and away again at different speeds (Figure 13). He plays with this for a while, and then puts his notebook, which is black, in front of the physical sensor. He notices the line is now higher than he has ever seen it before. He puts his notebook in front of the other reflectance sensor, and notices that its value jumps up to about 210. He removes the notebook, and the value returns to around 200 again.

Jim tells Christine that he wants to use the sensor values in a program, but is not sure how. Christine comes up with an idea: she wants to create a vehicle that roams around the room, away from the computer, and changes speed depending on the light conditions.

3.2.3. Programming an Autonomous Creature.

Once the *virtual Brick* is completely programmed, the program can be downloaded into the physical Brick. Then the Brick can be disconnected from the computer and the creature can be let loose.

The children remember from their previous experiments that the normal value of the reflectance sensors seems to be about 200, and ranges from about 190 to 210. Keeping in mind that the value range for motors is between -8 and 8, where negative numbers specify reverse, Christine suggests that they take the sensor value and subtract from it to map it into the range of motor values.

They open the rule editor for one of the motors and delete the value that is already there. Christine clicks on the left virtual sensor in the simulation environment, and they notice the name of the sensor, Reflect2, is automatically pasted into the rule. LEGOsheets uses the spreadsheet metaphor for building formulas in the rule editor fields by clicking on different cells in the simulation environment.

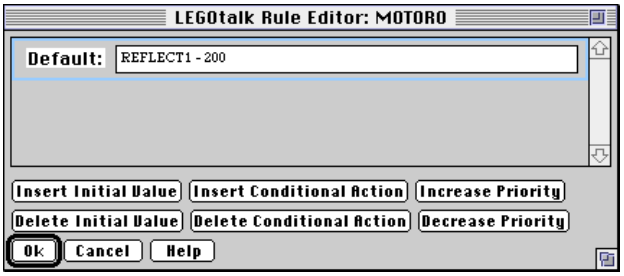

Figure 14: Rule Editor with Reflectance Sensor Value Used.

By subtracting different numbers from the value of the reflectance sensors in the rule and then running it while having the sensors wired, the children decide that the value 200 will work. They edit the rule for Motor0 as shown in Figure 14, and in a similar fashion, the rule for Motor1 except that the other reflectance sensor, Reflect2, is used. Once the rules are complete, they download the program to the Brick. Christine disconnects the Brick from the computer, puts the vehicle on the floor, and presses the start button.

At first, the vehicle does not move much, it just rocks back and forth a little. Jim remarks that it cannot make up its mind what it wants to do. Christine puts the white piece of paper in front of the vehicle, and it backs up a little. She moves the paper closer to the vehicle, and it backs up faster. She removes the paper from in front of the vehicle, and again it does not move much. Then she puts a black notebook in front of the vehicle, and it darts

toward the notebook. She quickly removes the notebook, and it stays still again. Jim grabs the white paper, puts it in front of the vehicle, and watches the vehicle back up. The children take turns placing the paper and notebook, as well as other objects, in front of the vehicle, watching what it does. Jim remarks that the vehicle likes dark objects and dislikes light-colored objects. Since their vehicle seems to behave like a living creature, they decide to call it their LEGOpet.

While exploring the behavior of the LEGOpet (Figure 15), Jim and Christine notice that when it runs into something black, it gets stuck because it continues trying to go forward. Since the LEGOpet has a touch sensor, they decide to modify the program so the LEGOpet will not get stuck.

Figure 15: Jim Playing with the LEGOpet.

After connecting the Brick to the computer again, the children open the rule editor for Motor0 and add a conditional action to the rule by clicking on the button labeled *Add Conditional Action*. They want to program the motor to go backwards at speed 8 when it hits something. Since they know the touch sensor returns 1 when it is pressed and 0 otherwise, they write the rule shown in Figure 16.

In the rule, the three sections after the word *If* specify the condition. The condition will be true when the touch sensor is pressed. The section after the word *Then* specifies the action to take if the condition is true, which in this case is to reverse the motor at speed 8. The section after the word *Default* is the default action, which will only execute if the condition above does not hold. Therefore, as long as the touch sensor is not pressed, the LEGOpet will act exactly as it did before. However, when the LEGOpet runs into an object, which will cause the touch sensor to become pressed, the motor will reverse at a speed of 8.

They repeat the same steps for the rule of the other motor, and then download the new program to the Brick so they can try it out. As they expected, the LEGOpet acts exactly as it did before, unless it runs into an object. When it does run into an object, it reverses for a split second, and then runs into the object again. The LEGOpet is never able to avoid the object again without assistance from the children. After watching this happen a few times, Christine decides the problem is that both motors are reversing at the same speed; therefore, the LEGOpet is not turning and cannot avoid the object. She tells Jim that they need to modify the program so it will turn when it backs up. She suggests using a value of -3 for one of the motors, and keeping the other motor at -8.

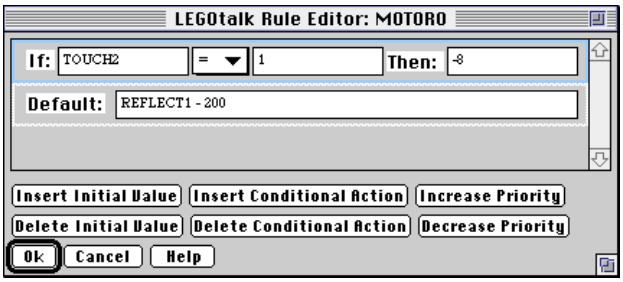

Figure 16: Rule Editor with Touch Sensor Used in Conditional Action.

Once Jim makes the suggested modifications and downloads the new program to the Brick, they start the LEGOpet again. This time when it runs into an object, it turns when it backs up, as they programmed it to. Sometimes it takes a few times of running into the object and reversing, but eventually it works itself clear of the object and continues on in a new direction.

4. Discussion

Through user testing, we have observed children having fun playing with motors using only the direct manipulation of the brick from LEGOsheets. Also, the ability to change the representation of the cell value from a number to a graph proved interesting to use in exploring sensor values, especially for the reflectance sensor. We believe the excitement the children experience while playing with LEGOsheets and the Brick in this manner creates an incentive that makes the step into programming easier.

The gradual introduction to the rule structure illustrated in the scenario shows that once children begin to program, they can reap the benefits of more powerful capabilities as they learn more about the rule structure. In the rule structure of LEGOsheets, interesting behavior can be achieved just by specifying Default actions; and if

more complicated behavior is desired, the user can make use of any number of Conditional actions.

An important aspect that we discovered during user testing is that children appreciate a lively application with colorful icons and audio feedback. Appropriate audio feedback serves two purposes. Users can receive confirmation that what they expected really happened or warnings that they just did something they were not supposed to.

The rule-based approach in LEGOsheets maps nicely onto programming artifacts with parallel behavior. However, it falls short in cases where the sequencing and timing of events is important. Brick Logo, with its sequential model, is better suited to handle sequences. A simple traffic light with a timed sequence shifting from red to green to yellow and back, can be written in a few lines of code that are straightforward. However, implementing the traffic light in LEGOsheets is more difficult, and not so intuitive.

5. Conclusion

The gradual transition from manual control to programmed control provided by LEGOsheets makes learning to program the LEGO Programmable Brick an enjoyable experience. It is important to make learning to program fun, especially for children. LEGOsheets achieves this by continuing to reward the children with increasingly powerful abilities while requiring only small increases in the skill needed.

Acknowledgments

The research was supported by the National Science Foundation under grant No. RED 925-3425, supplement to RED 925-3425, the Advanced Research Projects Agency under Cooperative Agreement No. CDA-940860, and Apple Computer, Inc. Special thanks go to Mitchel Resnick, Fred Martin, and numerous other MIT Media Lab members for the LEGO Programmable Brick and their great support. Scott Dixon, a teacher at the Centennial Middle School in Boulder, made all the user testing with his wonderful students possible. The people at the Center for LifeLong Learning and Design, provided essential insights and suggestions.

References

1. Braitenberg, V., *Vehicles: Experiments in Synthetic Psychology*, MIT Press, Cambridge, MA, 1987.

2. Hayes-Roth, F., "Rule-Based Systems," *Communications of the ACM*, Vol. 28, pp. 921-932, 1985.

3. Papert, S., *The Children's Machine*, Basic Books, New York, 1993.

4. Repenning, A., and W. Citrin, "Agentsheets: Applying Grid-Based Spatial Reasoning to Human-Computer Interaction," *1993 IEEE Workshop on Visual Languages*, Bergen, Norway, 1993, pp. 77-82.

5. Repenning, A., and T. Sumner, "Agentsheets: A Medium for Creating Domain-Oriented Visual Languages," *Computer*, Vol. 28, pp. 17-25, 1995.

6. Shneiderman, B., "Direct Manipulation: A Step Beyond Programming Languages," in *Human-Computer Interaction: A Multidisciplinary Approach*, R. M. Baecker and W. A. S. Buxton, Eds., Morgan Kaufmann Publishers, Inc., 95 First Street, Los Altos, CA 94022, 1989, pp. 461-467.

7. Resnick, Mitchel, “Behavior Construction Kits,” *Communications of the ACM*, Vol.37, No.7, pp. 65-71, July 1994.