Fun Learning Stagecast Creator: 
An Exercise in Minimalism and Collaboration

Cheryl Seals, Mary Beth Rosson, John M. Carroll, Tracy Lewis, and Lenese Colson 
Virginia Tech Center for Human-Computer Interaction, Blacksburg, VA 24060 USA 
{cseals, rosson, jcarroll, tracyl, lcolson}@vt.edu

Abstract

We are attempting to create a cross-generational learning community who will work together to design, construct, and discuss simulations of community topics. The simulations are built with Stagecast Creator, a state-of-the-art visual programming environment [12]. As part of this larger project, we have developed minimalist training materials for middle school students. This paper reports a formative evaluation of these training materials, in which groups of students worked together on two related tutorial modules. In general the students were successful in their work with Creator, needing little aid from the experimenters, and showing evidence of enjoyment. Our aim is to develop materials that will attract participation and enable students to spend their free time and play with this environment, and as a by-product of having fun, learn more about visual programming.

1. Introduction

Modern community networks leverage the resources and infrastructure of the World Wide Web to post and manage a variety of community resources. But it is not enough to simply post information once and for all: to ensure that a community network stays “alive”, its users must be able and willing to assist in maintaining its current resources, and to develop new activities that will engage each other and promote community interaction [9]. If this is to happen, community members must first acquire end-user programming skills, that enable them to design, contribute, and refine resources related to specific community concerns. We are exploring the use of one end-user programming paradigm that might be a source of community interaction—the design and discussion of visual simulations that illustrate community issues.

Our initial focus is on two very different segments of our community—middle school students and senior citizens. Our hope is that the simulation activities will bring these groups together, both in identifying and constructing simulation problems, and in discussing and refining existing simulations. An early design workshop with the senior citizens was promising, in that the participants quickly understood and “bought in” to the concept of community simulations, and they were able to contribute their own design ideas [9]. In this paper we report some of our early work with school children.

A critical issue has been how best to train and motivate students to participate in the simulation building activities. We want the students to be able to learn on their own, outside of the classroom, and to have enough fun doing this that they will continue to learn and to make their own contributions. As a starting point we have developed a minimalist tutorial for Stagecast Creator, the visual simulation tool that the students and other project participants will use to build their simulations.

In the remainder of the paper, we first overview the Stagecast Creator tool, and then describe our minimalist approach to training students to use this tool. In the balance we present and discuss the results of a formative evaluation of this tutorial.

2. Stagecast Creator

Stagecast Creator (SC) is based on a movie metaphor, where users create a cast of characters who interact and move within a simulation microworld in Stagecast Creator. This tool is a commercial version of KidSim/Cocoa, and includes the use of a macro-recorder to allow novice users to program by demonstrating an “example” [12]. Its
control structure is limited to a fixed sequence for testing rules. However, it embodies some important programming concepts, such as conditional execution, subroutines, iteration, and variables.

Users work on SC simulations using direct manipulation techniques [11] to program each character’s behaviors. The behaviors are represented as a set of “if-then” rules. A very common operation in SC is to create a rule. One or more preconditions is specified, and must be satisfied for the actions to be carried out.

To demonstrate a rule, the user indicates that s/he wishes to “create a rule”, and then selects the character being programmed. As shown in Figure 1, the Rule Maker window opens with the character shown in its current state. A bounding box or “spotlight” is resized by the user to allow enough room for the character to carry out actions, or to include other elements in the starting state. The user then enact the behavior they wish to see, for example dragging the character to a new position. All the demonstrated actions are recorded and will be executed when the character is in the appropriate starting state. The direct manipulation style of programming simplifies the creation of individual rules. However, considerable complexity arises when deciding how to distribute behaviors across characters, or how to prioritize rules within a character (only the first “matching” rule is activated on any tick of the clock during execution).

A character’s spatial context and visual appearance are required elements in a rule’s precondition: if a character is next to another while a rule is being demonstrated, both characters must subsequently be in that exact position for the rule to fire. Characters often have multiple “looks” or appearances. This appearance must be exact match for the rule to fire. For example, the starting state of the rule in Figure 2 shows a girl directly in front of a boy who is smoking a cigarette. Only in this specific situation will the rule be activated, such that the girl lights up her own cigarette. Having the girl in the right position is not sufficient; her appearance must also be that of the healthy non-smoking girl (Figure 3 shows other possible appearances). The dependence of SC rules on position and visual appearance cause visual brittleness that can lead to a computational explosion of rules when the number of possible scenarios is large.

From a programming perspective, another important characteristic of SC is its use of variables. In the example above, the characters are shown smoking in a schoolyard. As they smoke, an internal “sickness” variable is incremented. But in order for the simulation to make sense, this internal variable must be visualized, so that viewers can predict and understand its effects. In this case, the characters’ skin turns a sickly “green” to convey that they are getting sick, and eventually they collapse. The problem for novice programmers is to understand what an internal variable is, since it is not always visible, and how it can be used to govern characters’ appearance changes or other behaviors.
3. Minimalist Instruction for SC

The aim of our community simulations project is to engage people in creation and discussion of meaningful projects related to community issues [9]. We are seeking to build programming skills in a population of non-programmers, but we must assume that the SC learning and subsequent activity will be highly discretionary. These goals and assumptions led us to adopt a minimalist approach to training materials. In contrast, the interactive tutorial provided by SC is comprehensive, but introduces the functionality bit by bit, in very simple settings.

Minimalism emphasizes that people learn well by engaging in real activities; it encourages an "action- and task-oriented approach to instruction and documentation" [5]. It has been applied successfully to many learning situations, including word processing [5], desktop operating systems [5], hypermedia [1], and object-oriented programming [10]. Minimalist instruction attempts to help learners get started quickly, by offering them realistic tasks with meaningful goals from the start.

A key design challenge for our minimalist SC tutorial was to envision meaningful activities that would be engaging (for middle school students, our first focus for training), raise or discuss an issue of community importance, while also serving as a good vehicle for SC programming skills (e.g. rule creation and modification, characters and appearances, the use of internal variables).

We identified the topic of smoking in a schoolyard as something that we expected would have intrinsic interest to middle school students, and that raises issues of importance to the entire community (Figure 4). As part of a community simulation design workshop held prior to the work reported here, middle school teachers confirmed that kids in middle school experiment with smoking more than any other age group [9].

In the tutorial, learners initially simply run the example simulation to see what happens; this acts as an advance organizer for the guided exploration that follows. They then are asked to investigate the internal variable "sickness" that leads to changes in the smoker’s appearance: as the kids smoke, their level of sickness increases, causing their skin to turn "green", and eventually the character to collapse.

After exploring this behavior, the learners are guided to make changes to a rule, such that one kid to gets "sick" more slowly. They then enhance an existing rule, adding an intermediate appearance to the collapsing behavior.

In a second module, students are guided to create a new character (cigar smoke) that is based on the existing cigarette smoke character, but that has its own behavior as well. They also edit more of the existing simulation to make it more realistic, and add a new character, a teacher who "stops" the smoking. Table 1 presents a summary of the tasks included in each of the two tutorial modules.

Table 1: Minimalist Tutorial Tasks

<table>
<thead>
<tr>
<th>Activity I</th>
<th>Activity II</th>
</tr>
</thead>
<tbody>
<tr>
<td>• watch, answer questions about simulation</td>
<td>• reuse existing smoke to create new character, cigar smoke</td>
</tr>
<tr>
<td>• explore role of sickness variable in girl’s rules</td>
<td>• create new appearances suited for cigar smoke</td>
</tr>
<tr>
<td>• modify and test change of sickness variable</td>
<td>• edit a rule to use the new appearances correctly</td>
</tr>
<tr>
<td>• modify a rule to create a more realistic collapse</td>
<td>• add new appearances to smoke and use them to give more realistic look</td>
</tr>
<tr>
<td>• add a new character</td>
<td>• add teacher from drawer; walks, stops smoking</td>
</tr>
<tr>
<td>• give the new character a movement rule</td>
<td></td>
</tr>
</tbody>
</table>

Throughout the design of the tutorial, efforts were taken to minimize the amount of reading and passive observation expected of students, and to provide open-ended questions that stimulate exploration. To help learners avoid or recover from errors, images of the proper ending state for many programming steps were provided, and checkpoint explanations and tips for error recovery were inserted at key points. In general, the minimalist character of the tutorial can be summarized as follows:

- Learner’s first activity is exploration of a full-size simulation.
Learners immediately "dive into" the example, investigating a character, its internal variables, and its behavior.

To reduce errors, learners are given a visual image of the target state for a task.

Text is minimized by providing only essential explanations, and by use of pictures

Details of a procedure are only given once: if the learner is asked to repeat a similar task, they are expected to remember or infer the steps needed.

This work reports a formative evaluation of the tutorial, in that our main goal was to understand its strengths and weaknesses. However, in support of our higher-level project goals, we were also curious as to what the students would learn about SC and its visual programming paradigm through this brief tutorial. Thus we also prepared a Knowledge Survey to be administered individually at the end of the tutorial session.

The knowledge survey was designed to be a high level self-assessment of the learning experience and participants' current understanding of key concepts. The first portion of the questionnaire measured subjective reactions to SC, as expressed on a five-point Likert scale of agreement with these statements: SC is easy to use; Directions were hard to follow; SC is boring; I understand SC well; I would like to use SC in my classes.

The second portion of the survey attempted to assess learners' understanding of internal variables, and of the role of a character's visual and spatial context. We probed this by presenting visual situations and asking the learners to make predictions about what would happen given a rule, a variable value, and so on. We tried to make these questions as near to the learners' experience as possible by using Smoking-Kids scenarios similar to those encountered during the SC tutorial.

4. Procedure

Participants included ten middle school students—nine boys and one girl—who were recruited through a hand-out provided to several teachers at the middle school. The SC tool was described briefly, and students were told that they would receive a small stipend ($15) for participation.

Students were scheduled in pairs (although in one case, only one student arrived, and in another there was an extra; see Table 5). Each participant completed a brief demographic survey that asked him or her about their prior experience with computers in general, as well as with video games or simulation software.

At the start of the tutorial, participants were asked to designate one person as "driver" and the other as navigator; they were also encouraged to exchange roles halfway through. They were asked to think aloud while working through the tutorial together; this facilitated their collaboration, while providing insight into their goals, plans, and concerns [8]. During the experiment, the evaluators noted positive and negative critical incidents (e.g. documentation, confusion, wrong path, general errors, and just in need of help.) These critical incidents would indicate where SC usability became an issue, problems with the tutorial as written, or visual programming concepts that seemed to be particularly difficult.

The study was conducted in a usability testing room; session video, audio and screen were recorded for later review. The evaluators observed through a one-way mirror, and were able to use an intercom system to provide assistance if necessary: they followed a policy of minimal interaction, hoping that the students would reread the materials if they got stuck. If problems arose, the students were directed to reread their materials, so that explicit help was offered only when the students became very perplexed or frustrated.

After the experiment, participants completed the Knowledge Survey. They were also given an informal exit interview to ascertain if they understood all the questions in the Knowledge Survey, if they understood the basic message in the tutorial (that smoking is bad for you), and finally to gather suggestions for future simulations.

5. Observations and Results

Participants spent a range of 30-55 minutes working through Activity I and 34-47 minutes in Activity II. All participants had previous computer and drawing experience. In general, most of the students were quite successful and engaged with the SC environment and with the tutorial tasks. However, this general success emerged in spite of various problems, some specific to individuals, others more general. In the following we discuss the most interesting of the problems and successes we observed.

5.1 SC Usability Issues

In the Knowledge Survey and exit interview, students seemed to feel that SC was easy, fun to use, and that they
would like to use it in their classes; but they also indicated that they needed more exposure to feel confident of their understanding of the environment (see Table 2). Thus while we feel we are on the right track with these training materials, we clearly need to provide more help.

### Table 2. Subjective Reactions

<table>
<thead>
<tr>
<th>Likert Agreement Scales</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stagecast is easy to use</td>
<td>2.5</td>
</tr>
<tr>
<td>Directions were hard to follow</td>
<td>2.625</td>
</tr>
<tr>
<td>Stagecast was boring</td>
<td>4.125</td>
</tr>
<tr>
<td>Understand Stagecast well</td>
<td>3.125</td>
</tr>
<tr>
<td>Like Stagecast to be used in my class</td>
<td>2</td>
</tr>
</tbody>
</table>

5-point scale from 1=Strongly Agree to 5=Strongly Disagree

In observing their interactions with the SC tools, it was clear that the students had no problem with the appearance drawing tool (a bit editor with brush and color controls, etc.). We suspect that this is due to their general familiarity with computers and drawing tools; for example several of the students were already users of PowerPoint.

SC is very colorful and inviting, and comes with many demos of a game-like quality. One group commented that they felt they were making a video game. Its character appearance editor has functionality similar to common drawing packages, yet adds features that are helpful for visual programming like rotation and stretching a character. Consequently, none of our users had any problems with drawing characters. They were also able to use the programming by demonstration (PBD) mechanism for making new rules.

However, students experienced difficulty at times when managing the many small tools provided by SC for building or editing behaviors. Often the learners would have the system in an incorrect mode or would be addressing their input to an inappropriate window.

For example, one of the subtasks in Activity I was to introduce a more realistic look to the "collapse" rule, namely to show the girl in a sitting position prior to collapsing onto the ground. The solution involves inserting a new appearance (pre-drawn in this case) into the actions carried out at "collapse" time.

In order for the collapse rule to fire the girl must be holding a cigarette and showing her "sick" (i.e., green) appearance; her sickness variable must also be greater than or equal to 12 (Figure 5a). When these conditions are met, the actions in the post condition are to be executed: the girl will sit-puff, sit-hold, and collapse as a 3 step animation (see Figure 5b).

![Figure 5a: Preconditions of Collapse Rule](image)

![Figure 5b: Actions of Collapse Rule](image)

Initially, the actions of the collapse rule included only the last action shown in Figure 5b, so the learners were guided to animate the sit-puff and sit-hold action and then insert them into the right position (rule-ordering exercise). All of them understood how to specify these new appearances, but had considerable difficulty determining or ensuring that they were in the right order. In particular, we noted confusion at this point between the rules listed in the character window (where the collapse rule is just one of several), and the list of collapse actions shown in the rule editor. The tutorial was clear on which tool the learners should focus on, but rather than reading this material, they tended to simply select the previously active tool, directing their focus away from the detailed editing of the collapse rule. We suspect that this was due to a combination of wanting to act rather than read [6], and that the character window had been their main focus up to this point.

Table 3 summarizes this and other problems that we attribute to the usability of the SC environment. The environment "hides" functionality in drawers that are
opened and closed with orange tabs. This gives the environment a simpler look at times, but requires the users to learn the meaning of several similar but generic interface icons. The metaphor of some of these icons and too much scaffolding of the environment caused problems. It also means that the materials for many tasks are distributed across many small “helper” windows, leading users to open many different (often similar) windows in the course of creating or modifying a single rule. This window management problem typifies a pervasive concern with rich direct manipulation environments like SC; designers seek to simplify and manage complexity by decomposing and hiding functionality, but discovering and using features distributed in this fashion causes its own problems.

Table 3: Stagecast Usability Problems

<table>
<thead>
<tr>
<th>SC Usability Issue</th>
<th>Likely Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directing input to the wrong window</td>
<td>Too many similar-looking windows</td>
</tr>
<tr>
<td>Confused between rules and rule-actions lists</td>
<td>Lists that look similar but different meanings</td>
</tr>
<tr>
<td>Select wrong orange tab</td>
<td>Multiple similar icons</td>
</tr>
<tr>
<td>Trouble finding the “show the tests” subpane</td>
<td>Icon labels appear after hovering a few seconds</td>
</tr>
<tr>
<td>Inability to find rules or other content in windows</td>
<td>Non-traditional method of scrolling</td>
</tr>
<tr>
<td>Misunderstand spotlight and concept of stretching it</td>
<td>Spotlight metaphor is not obvious or intuitive</td>
</tr>
</tbody>
</table>

Other problems can be traced to specific user interface choices made by SC designers. For example, the scrolling within a rule or action list is done by clicking on arrows at the bottom or top of the list (which also signal that there is “more” in that direction). This does not map well to people’s general familiarity with scrollbars, and not surprisingly we observed problems in its use. The many icons provided for accessing system features have “tooltips” that appear when the mouse pointer pauses over them, but these users were not familiar with this technique. In our next iteration, we will more carefully introduce and motivate these user interaction details.

While the learners seemed to have no problem with the general concept of demonstrating and animating actions to create rules, there were some aspects of the movie metaphor that were less effective. The most salient was the “spotlight” concept, wherein users drag a yellow box to specify the starting spatial container for a rule. The box can be dragged in all four directions to size it, and it must be enlarged enough to include all visual preconditions (e.g. a nearby character). Of course, problems with this feature may reflect deeper problems with the SC visual programming paradigm—what seems obvious in a user’s view of the simulation (namely its current visual state) must be specified explicitly as part of rules that control it.

5.2 Challenges of Visual Programming

Some of the problems experienced by students highlight general characteristics of visual simulation programming that may be problematic for novice programmers. One of these is the inherent relation between internal variables and the visual state of the simulation. In SC, global and character-specific variables add powerful functionality in terms of tracking and modifying a simulation’s state. As a result we emphasized the use of variables in the sample simulation. However, some students had trouble grasping the logical relation between a variable and the visual state apparent on the screen.

For example, in one case, the tutorial asked the students to look at a rule and explain how “sickness” is tested. In this case, there was a test that determined whether the variable was greater than or equal to 10. But in looking at the tests, one pair seemed to confuse the variable and its value with the character’s appearance (which in fact changes as a consequence of changes in this variable): One learner begins, “The sickness is tested by the …”, and her partner continues “… appearance of the character”.

Another problem that stems from the visual character of SC programming concerns the role of visual context in the before-after rules. As emphasized earlier (see Figure 2), the characters must be in exactly the right position and exhibiting the correct appearance in order for a rule to fire. Often the students seemed not to understand how important it was to get this context exactly right. For instance when asked to create a cloud and make it move, one student positioned it on top of another character (the basketball hoop), because he liked the way it looked. The problem with this is that a rule demonstrated for the cloud now includes the basketball hoop as part of its visual context, making the rule too specific to be useful.

5.3 Minimalism

All of the learners were quickly engaged by the SC activities; most began to explore on their own within the first few minutes of their tutorial. At the same time, they were able to check back with the tutorial to find new suggestions for activities, which kept them moving forward and learning new skills. Thus in general we are pleased...
with the active and exploratory learning that the tutorial seemed to encourage.

The minimalist strategy of anticipating likely errors and inserting recovery information also seemed to work. While students got “off track” often, they almost always were able to either recover on their own through further exploration, or use the tips provided in the tutorial (e.g., resetting the simulation became a common “fix”). There were only two instances in which the experimenters needed to provide assistance: one was the singleton learner, who was unable to answer one of the questions posed by the tutorial and began repeatedly looping through a set of actions (the experimenter advised him to simply continue). The second was a pair who had trouble re-ordering a rule’s action list. These difficulties will be addressed directly in our next iteration.

Details of tasks were only given once, but our users seemed to have no trouble completing tasks a second time on their own. Our attempt to minimize the text by inserting icons and screen shots seemed to work well for these users: they often skipped the text to find the next visual instruction to mimic. This enabled rapid exploration, but at times caused problems (e.g., when they had to come back and re-read to find key bits of explanation).

5.4 Collaboration

When planning this study, we decided to study students working in pairs, because we thought it would be a more natural context for gathering the “think aloud” data (i.e., discussing goals, actions, problems). But as we carried out the study, we were struck by how much the collaborative setting added to the learning experience (see also [7]).

Our initial plan was to observe five pairs of students, but one person showed up on the wrong day and worked alone; we added his intended “partner” to another pair, making one group of three (see Table 5).

In general, groups 2, 3 and 5 seemed very cooperative, with very good coordination between navigator and driver. Group 2 is perhaps a paradigmatic example: in this case, one partner was a female, who made suggestions and provided explanations for her partner who was “driving”. He tended to just try things, but she was able to help him understand what had happened and what to try next by referring back to the tutorial and finding relevant explanatory material. This is an interesting approach to the well-documented problem of coordinating training materials with system state [5]. If two people learn together, the “non-actor” can take on the responsibility for finding useful information in the written materials. Group 4 (P6, P7, P8) seemed to have the most “fun”, going well beyond the tutorial tasks in their exploration. The minimalist tutorial worked particularly well for them: they jumped around in the tutorial, skipped steps they could infer from prior knowledge, and still managed to accomplish the tasks more quickly than other groups (64 minutes for both activities). With their guided exploration play session they created very active simulations a little more so than specified in the tutorial documentation. This leads us to believe that if we can instigate episodes of “play” guided by a minimalist tutorial, students may be able to learn SC without assistance.

Table 5: Group Composition and Collaboration Style

<table>
<thead>
<tr>
<th>Group</th>
<th>User/Gender</th>
<th>Learning preference</th>
<th>Collaboration Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>P1/M</td>
<td>Art/drawing</td>
<td>N/A. P1 was frustrated especially when having to complete tasks where some of the key areas were initially scaffolded from his view</td>
</tr>
<tr>
<td>2</td>
<td>P2 M, P3/F</td>
<td>Science/Math, Art/drawing</td>
<td>Cooperative. One member of the group was very inquisitive, the other member was pragmatic and kept the group on task</td>
</tr>
<tr>
<td>4</td>
<td>P6/M, P7/M</td>
<td>Both equally</td>
<td>Guided exploration &amp; play session. The group leader “navigator” was most aggressive person and he worked well with the driver. While the passive person of the group contributed very little to the interaction, we tried to prompt him to contribute, but he remained passive</td>
</tr>
<tr>
<td>5</td>
<td>P9/M, P10/M</td>
<td>Both equally, Science/Math</td>
<td>Very Cooperative. Very good coordination between navigator and driver.</td>
</tr>
</tbody>
</table>
5.5 Knowledge Gained

In the Knowledge Survey, participants were asked several questions designed to probe their understanding of SC programming.

One programming concept introduced during the tutorial was that of a variable— the "sickness" variable plays an important role in the simulation, determining when the characters change color, collapse, and so on. Thus, one question in the test asked them to explain what happens in Figure 6, which shows a before-after rule and a change in the sickness variable.

Most participants understood the general connection between the character’s visual appearance and the value of the sickness variable. For example, answers to this question included:

- “First she dies and then she gets back up”
- “The girl collapses and then she recovers in the after”
- “She got sick and collapsed, then reset”
- “Her health returns and she gets up”
- “Sickness level goes down from 14”
- “If she has collapsed and her sickness returns to 0 she gets better and stands up”

All of these responses suggest a basic understanding that sickness is a controlling factor in the girl’s behavior. Some expressed this in a rather qualitative way, but the last two responses above suggest a more “algorithmic” understanding of this variable and its consequences.

The Knowledge Survey questions causing most difficulty were those that probed learners’ understanding of the visual state specified in a rule. For example, they were shown Figure 7 and asked if the boy would begin to smoke in this situation (the first-puff rule requires the other character to show its “healthy” appearance, see Figure 2). Only two participants recognized that the before part of the rule “the actual color and position of the girl” mattered and must match exactly in order for the after part “the boy’s first puff” to be carried out. This is a crucial element of SC rule formation and needs to be emphasized more in future training materials.

6. Discussion

The minimalist tutorial for SC was generally successful in exposing students to the basics of SC programming. All participants were able to successfully work with the environment and complete simple modification tasks. In most cases our participants easily recovered from errors and were able to skip through the materials, interacting with the system and the written text in an interleaved fashion.

With our Knowledge Survey, we were able to identify that students had problems with matching the visual state of the rule and adhering to the exactness of visual syntax. This is an inherent problem of visual languages that depend on programming by demonstration techniques. PBD is a great technique to get novice users started quickly, but is very limiting when users want to create a series of sophisticated behaviors.

Also we have identified that these types of languages are very brittle when it comes to the reuse of rules; whenever a user wants to create a simple variation on a rule, s/he will have to recreate all the rules for this variation which can become very tedious for more sophisticated scenarios. Also this work indicates that using a production system, where the first rule that matches the situation is fired in order, simplifies the system, but causes a problem that only the first rule will fire if it is always matched, which may lead the user to believe that none of their other rules work. A system that is based on object-oriented techniques may remove this confusion. Instead of having to use a pseudo non-determinism hard coded into our simulations, having to order the rule specifically, we would prefer the characters in our simulations to be able to interact and respond to each other with message passing. With this strategy a
character would react when requested. This will allow a simulation to be a collection of characters or objects that work collaboratively, and allow greater reuse of a character’s behaviors.

This formative evaluation also illustrated the challenges of developing minimalist self-study materials for a highly visual, direct manipulation environment. At any given moment, SC presents many options through icons or other visual elements. The opportunities for exploration are many and varied, and these students were quick to explore. Nonetheless, the evaluators were required to intervene very little, and three of the five groups required no help at all. This leads us to believe that minimalist training materials and collaborative learning may be a good combination.

We chose to study student pairs for methodological reasons: to make the students feel more comfortable and to encourage them to share their thoughts. But we were impressed with how much the collaboration added to the learning experience, especially when trying to recover from errors, and in generating explanations for surprising events. This is clearly an example of constructivist learning, where the students were learning through exploration, discussion, and the development of shared understandings [7] [13]. But beyond the cognitive assistance they provided in keeping each other on track and in understanding what was happening, their collaboration seemed to make the learning more fun.

As a part of the larger community simulations project, our findings are encouraging: it seems that students are willing and able to use this tool to work on community-related simulations like the Smoking-Kids. Indeed they seemed to enjoy the process. They also were able to supply ideas for future simulations: sports teams, computer games, teenage flirting, and so on. This suggests that in the longer term they may be able to contribute their ideas to a repository of simulations [9].

7. Future Research

We are considering ways to promote collaborative learning more generally, as we recruit additional students and other community members into the project. Investigating gender issues may also be interesting, especially since we could only get one girl interested in participating in our experiment, and the pairing that included the girl made the best use of the minimalist manual by carefully utilizing the help it provided. Our subjects were self-selected volunteers, and we can only suppose that boys are very interested in computer programming and computer game type activities.

The students we recruited were quite successful with the SC tutorial, but we did identify a number of issues that must be addressed by future work. Some of these were basic issues related to the visual programming paradigm, for example the relation of internal variables to the visual state of a character, or the role of specific visual context in creating rule preconditions. Other problems were more specific to SC, such as confusion about icons or management of many small and similar windows. We also identified opportunities for improved our tutorial with improved error detection and recovery information, for instance when guiding the learners in re-ordering actions. The next iteration of the tutorial will attempt to address these learning problems.

Our next step is to perform a comparative analysis between our minimalist materials and the provided systems approach materials, in an effort to gauge whether minimalism offers a better medium for training, and enhances user learning and enjoyment of the activity. We will also increase the scope of our empirical testing with a user group of senior citizens. Locally this group is very active and many have time to devote to exploratory study, and to mentor our students and help them refine their simulations.

Our goal with this cross-generational learning community is to have people from segments of the population that would not normally collaborate to actually work with one another and get excited about and have fun with simulation programming. We want to use the experience of our seniors about civic events to aid students with realistic design of simulations. This project could spark discussions within the community with groups that rarely interact by identifying individuals to work together to design, construct, and discuss simulations of community topics.

8. Acknowledgements

This work was supported by a grant from the National Science Foundation (NSF ITR 0091102). The authors are members of the Center for Human Computer Interaction, Department of Computer Science, at Virginia Tech. We would like to thank other members of the Community Simulations project, for helping to perform this study. We would finally like to thank area teachers for identifying students, and students from area schools for their participation. Finally, we thank Stagecast Creator for permitting experimental testing, review and critique of their software.
References


