Can Middle-Schoolers use Storytelling Alice to Make Games? Results of a Pilot Study

Linda Werner
Assoc. Researcher & Lecturer in CS
1156 High Street MS: SOE3
UCSC, Santa Cruz 95064
001 831 427 2076
linda@cs.ucsc.edu

Jill Denner, Michelle Bliesner, & Pat Rex
ETR Associates
4 Carbonero Way
Scotts Valley, CA 95066
001 831 438 4060
jilld, michelleb, patr@etr.org

ABSTRACT

In this paper we share experiences from two 2-week summer courses for middle-school students in game programming using Storytelling Alice (SA). The students spent 20 hours learning SA and creating their own 'games' alone and in pairs. We discuss problems and preliminary findings regarding game programming by middle-school students. Our findings suggest that middle-school students can use SA to make games, and that this activity can be used to build information technology fluency.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education–*computer science education, literacy*

General Terms

Languages.

Keywords

Middle-school, pair programming, Storytelling Alice, IT fluency.

1. INTRODUCTION

The ubiquitous use of computers and other electronic devices by middle-school students builds many aspects of information technology (IT) literacy. However, in order to adapt to rapidly evolving information technologies, today's students need to be more than IT literate. Learning and working in all disciplines now requires them to be IT fluent; that is, they need to develop the capacity to think systematically and creatively with technology. In 2007, the International Society for Technology in Education released the National Educational Technology Standards [11] about what K-12 students should know and be able to do with technology. They emphasize the importance of creative thinking and innovation - using technology to learn rather than learning to use technology. They also describe the importance of critical thinking, problem solving, and decision-making using digital tools. But the related concepts and skills (algorithmic thinking, programming, modeling, and abstraction) are generally taught only in computer science courses, which attract a limited number and type of students, and are rare in K-12 settings. We believe the creation of IT, not just the use of IT, provides the opportunity for a range of students to make invaluable IT fluency gains for these fundamental concepts, and many other IT fluency aspects [18].

In 1999 the National Research Council (NRC) defined IT fluency [16] by taking IT literacy and adding to it the ability to independently learn and use new information technologies as they develop. Their definition of IT fluency also states that algorithmic thinking, programming, modeling, and abstraction be used to solve problems because these concepts provide the basis for students to build their understanding of new information technologies. These four concepts are fundamentals of any computer science curricula and are basic to most introductory programming courses at the high school and university level. The widely used NRC model of IT fluency was developed for college students, so little is known about what IT fluency looks like in middle-school.

The Computer Science Teachers Association in its Model Curriculum report [4] makes suggestions about appropriate topics and goals for the middle-school years. They state "(w)e believe with teachers who believe that students at this age ought to begin thinking algorithmically as a general problem-solving strategy… Thus it makes sense to develop more teaching strategies that encourage students to engage in the process of visualizing an algorithm. Seymour Papert's pioneering experiments in the 1980s corroborate the belief … of how K-8 students can be engaged in algorithmic thinking" (page 9). However, the 6-8 grade level breakdown does not specifically identify outcomes for engaging students with algorithmic thinking, modeling, and abstraction and these topics do not enter the grade level breakdowns until the high school years.

Previous research suggests that making a game may engage students in IT fluency by increasing motivation and engaging students in critical thinking and problem solving. The motivation comes from the popularity of gaming and the excitement of making a unique game that others can play. Others have found that students are motivated to problem solve in order to make their games meet their design goals [17]. According to Gee [10], in the modern age, we need ways to help students "think like game designers." Salen [19] describes what this looks like when she says, “Knowing how to put together a successful game involves system-based thinking, iterative critical problem solving, art and aesthetics, writing and storytelling, interactive design, game logic and rules, and programming skills” (p. 305). Game programming challenges students to think in certain ways: as opposed to multimedia design, it offers opportunities to engage in a representational form of computer literacy [9]. Making games requires students to make explicit their assumptions about what makes a good game [13] because they need to think about how the player will interact with the game, the outcomes of player action, and the goal of the game. By goal of the game we refer to how the player can win or lose the game.

For six years we have been involved in teaching middle-school girls and boys to create games using pair programming. Our research on pair programming at the university level has been recognized by the National Center for Women in Information Technology as helpful for the retention of women in university computer science programs [2]. Our findings suggest that game design in pairs has real promise for promoting IT fluency in middle-school, and we have published the results of that work in several places [3, 5, 7, 8, 21, 22, 23, 24]. However, there were limitations to the programming environments that were used. We found that Flash with actionscripting was too difficult for this age group to use independently and Stagecast Creator was not visually appealing to the students because the animations, videos, and games they created were only 2-dimensional.

We decided to extend our research on game creation by using Storytelling Alice (SA) because it offers opportunities for students to engage in key programming concepts while creating very appealing 3-dimensional movies, stories, and games.

Storytelling Alice (SA) is similar to Alice, but was designed to engage younger children by offering higher level animations [15]. There are additional characters (e.g., fairies and typical school students) and additional pre-programmed behaviors (e.g., dancing, kissing, and talking) that appeal to middle-school students.

During the summer of 2008, we ran two courses that were not part of a controlled study, but instead were part of an NSF-funded feasibility study to determine if middle-school children can use SA and pair programming to create computer games. Specifically, we were looking for answers to the following questions:

- Can SA be used by middle-school students to make games?
- Does game programming with SA support the development of IT fluency?

2. METHOD

2.1 Participants

During the summer of 2008, we conducted courses for middle-school girls and boys in game programming using pair programming and SA. These two 2-week courses ran simultaneously for two hours each weekday; one was located at a community center in a small city surrounded by a large agricultural area, and the other was run at a Boys and Girls Club in a small city. Participation was voluntary and free, and students were recruited by club and center staff from existing membership lists and nearby schools.

We started our courses with 43 student participants and two seasoned middle school teachers. Of the 43 starting participants, 33 participants completed the pre-course survey. Twenty-eight participants completed the course, and 22 participants completed both the pre-course and post-course surveys. Table 1 gives details of these 22 participants.

Table 1. Details of Participants

Boys	12	
Girls	10	
Mean age	11.5 years	
Age range	10-13 years	
Modes	11 and 12 (8 each)	
Grades entering next year	6th	11
	7th	9
	8th	2
English spoken at home	Only	6
	Mostly	2
	Half of the time	13
	Infrequently	1
Most frequently spoken at home other language	Spanish	
Ethnicity	Hispanic or Latino	15
	White or Caucasian	6
	African American	2
	Didn’t answer	1
Ever written a program before this course	Yes	2
	Never	15
	Didn’t know	3
	Didn’t answer	2
Computer at home	Yes	14
	No	7
	Didn’t answer	1

2.2 Procedure

One of the instructors is a regular classroom teacher with many years of experience using technology instruction, but little prior knowledge of SA. The other instructor had prior SA and Alice knowledge and is a teacher of technology classes to grades K-12, including animation, computer literacy, digital storytelling, digital photography, electronics, and clay animation. Their training for our program involved a two-hour meeting in which they learned about the research goals and the instructional approach, as well as four hours of self-directed learning of SA by doing built-in tutorials and the first six of the challenges (SA exercises we created for the students).

Classes met daily for two weeks, two hours each weekday. During the first five hours of class time, students were assigned temporary partners. Together, they played sample games and used other SA programs, participated in a discussion about the qualities of the games they played, and worked through four tutorials that come with SA. For pair programming instruction, the teachers acted out examples of successful and unsuccessful pair programming.

On the third day of classes, most of the students were assigned to their permanent partners. We paired them based on a list of names of at most three others they'd prefer to work with. In most instances we were able to pair a student with one person from their list. Once paired, the students started working on exercises we call 'challenges.' We based our challenges on those designed primarily for use by high school and college students learning Alice by Shelley, Cashman, and Herbert [20]. We adapted existing challenges so they could be used in SA by a younger audience and created additional challenges that focus on game elements. These challenges were formatted as check lists and organized so that students gradually moved from following directions to making their own design decisions as they mastered each new programming skill. During hour seven we discussed the concepts of storytelling and demonstrated storyboarding. The participants then worked additional challenges covering aspects of events and event handlers. During hour 11, the students sketched their own stories on a single sheet of paper and started programming them. They then turned their own stories into 'games' based on group discussions of 'gameness' qualities including aspects of user interfaces (gameness is defined in more detail below). The students spent the rest of the class time (8 hours) creating their own games and working on challenges for concepts the students identified they needed for their games such as subtleties with event handling, if/else, looping, counters, and timers.

We found students preferred to have as much hands-on computer time as possible. Consequently, we modified our plans for the course to accommodate this, creating more challenges and removing teacher-led instruction. Comments from students include,

- *"The challenges were fun. They helped me do most of my game and do questions." (male student)*
- *"(What I like best about SA) are the challenges. We had to figure it out. I like doing that." (student)*

2.3 Measures

We used both quantitative and qualitative data collection methods to learn about the participants and the results of our courses. We administered pre- and post-course surveys to obtain information about the participants, such as demographics, experience using computers, career plans, and understanding of SA. Some statistics from those survey questions are given in the 'Participants' subsection of this section. Qualitative data were collected through several means both on a daily basis and at the end of the two-week period. Each day of the program, one project team member at each of the two sites recorded observations made during class using an observation protocol. In addition, teachers recorded activities, objectives met, and emerging issues using a daily log template. At the end of the program, project team members and teachers discussed the project's research questions during a team interview; one of the PIs conducted one-on-one interviews with both teachers; a project team member interviewed a teaching assistant from one of the sites; and three project team members conducted individual interviews with a total of 11 students from five pairs and one solo (six interviews at the community center and five interviews at the Boys and Girls Club); five of the students interviewed were girls, and six were boys. In addition, computer-logging data were used to capture students' actions while using SA. These data were collected every day and were automatically saved in files on their computers. In addition, the 23 programs identified by the students as their best creations were coded for qualities of 'gameness' and aspects of IT fluency.

3. CAN STORYTELLING ALICE BE USED BY MIDDLE-SCHOOL STUDENTS TO MAKE GAMES?

We chose SA because we believe this age group can easily learn it, it is fun to use, and it can be used to create 'games.' However, we found that several factors need to be in place to optimize game production. In particular, teachers need enough time before the course starts to learn SA, 'work through' all the challenges, and hone the skills that the students will be learning so they can show the students (especially the ones who advance quickly) how to troubleshoot. Students with a range of skill levels thought of creative ideas for games, but not all could be implemented due to limitations in teacher knowledge, as well as time limits in the class. Since it was not possible to anticipate students' interests before the class began, several new 'challenges' were created mid-course by project staff in order to guide students on how to carry out certain ideas about their games.

In general, the teachers felt that the challenges were a good strategy for students to learn the elements of SA they would need for their games. In a post-course interview of the teachers, one teacher described the course's game development process in the following way,

> "…we structured the class so that they would, first, do the challenges and then make a game. So that's exactly what the game was. [It] was taking what they learned and the challenges and using it, changing it. One of the choices that we gave them was to make their game from an existing challenge, I guess altering the challenge. I only saw one person, … who used the setup of the… amusement park to make his game. Everybody else… did their games from scratch. So they didn't actually change the challenges, they were more like taking everything they learned and using it to… build their game."

The teachers also thought that SA was challenging, but the class provided the right level of resources to allow students to problem solve on their own. For example, when asked what students did if their program did not work as they expected, one teacher responded:

> "They usually changed it to work as they wanted. They would usually troubleshoot it and… talk it… with each other and try to figure it out. Sometimes they would ask other students, sometimes they would ask me or sometimes they would go back to a challenge and some of them actually went back to tutorials to learn them. But generally, yes, they wanted to always make it do what they wanted..."

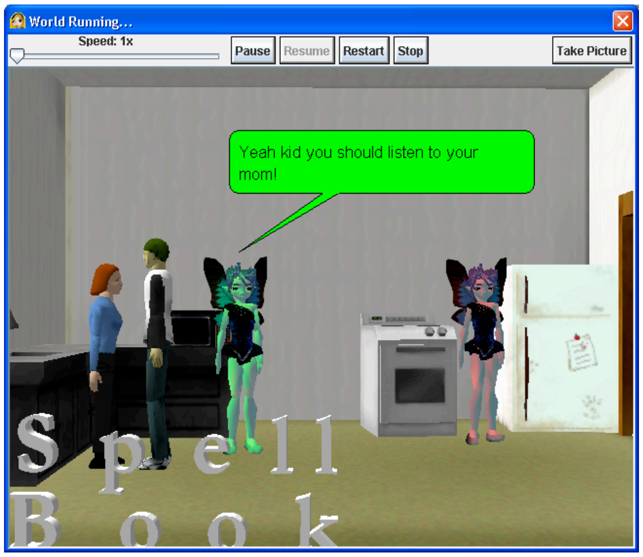
Although the teachers believed that SA was a good platform to use for game programming, we wanted to measure the extent to which the students' projects meet outside criteria for games. To this end, we developed a 'gameness' coding scheme based on Juul's six defining qualities for games: existence of rules, different outcomes that can be quantified, the presence of a goal, the player uses effort to effect the outcome, the player is attached to the outcome (i.e., happy with positive outcome, unhappy with negative outcome), and the rules can be played with or without real-life consequences [12]. We also drew on Costikyan's definition of games, which says, "without a goal, it is a 'toy.'" A game is "a set of player-defined objectives overlaid on the toy." Costikyan also says that games are forms of art with emotional appeal [6].

None of the students' programs have real-life consequences, so we ignore that quality. Below we describe the remaining *five defining qualities* and give statistics for each of them for the 23 programs identified by the students as their best creations. The two PIs each coded the programs separately, discussed their results, and then came to an agreement to reach 100% inter-rater reliability.

- A game has rules, meaning the game requires user input, correct input is described, and player actions result in the state of the game changing. We found that 100% of the programs have rules; however, seven programs (30%), have faulty mechanisms used for explaining the rules to the player. For example, rules may exist in the SA program code and may serve as instructions to the player, but are not given in their entirety to the player or the rules are inconsistent with the instructions given to the player. For one of the programs a very sophisticated mechanism is used for instructions: the player needs to discover and click on a 'spell book' for the instructions to be displayed. A screen shot from this program called 'Invisiboy' is shown in figure 1. In 'Invisiboy', the fairies carry the boy away because he has not listened to his mother. The rules for his movement are displayed only if the player can find and click the 'spell book'. Figure 2 is another example of game rules. The pair of students who created the program called 'Hurry' provide instructions to the player in a display that sits below the 3-dimensional world: "Click on the fish so you can catch them but if your time runs out you loose (sic)." The player is not told how many seconds the play can continue. The students were experimenting with using different values for the timer and did not have sufficient class time to include that information in their program. When the time runs out, the player is told how many fish are caught and whether he or she has lost the game.

- A game has a goal, meaning the game has a clear beginning, middle, and end; and the objective is clear to the player. Thirteen programs, 56%, have goals. In one of these, the pair of students could not figure out how to create their program to allow players to meet their goal; however, 12 programs contain the code to allow the player to meet the defined goal. As mentioned earlier, the students clearly described the goal for the player in the 'Hurry' program, shown in figure 2. The goal is to catch all the fish before the time runs out. If the time runs out before all the fish are caught, the player loses the game. The player is told with the display of "LOSER" in large letters if fish remain to be caught (see figure 3).

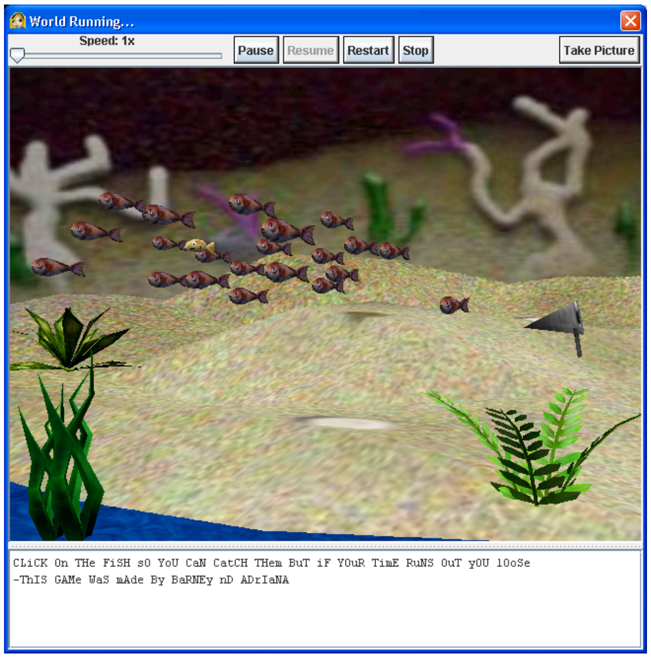
Figure 1. Invisiboy Screen Shot



Yeah kid you should listen to your mom!

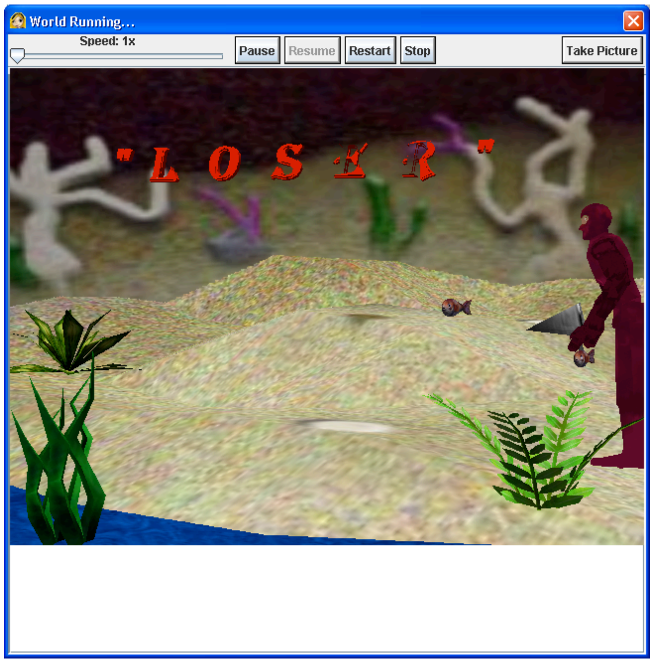
- A game's outcome is uncertain, meaning there are different possible endings, and there is a clear way to reach the player's goal. Fourteen programs or 61% allow for different endings. Examples of these are: characters are eaten or not, a dog is reconnected with its tail or it isn't, and robots are discovered or remain hidden, as seen in the program 'Find the Robots' in figure 4. In this program, the only rules that are displayed are spelled out by the small green letters that give the name of the program. The robots are initially hidden behind the larger letters with only a small part of each of the robots visible. If the player sees and clicks on the visible part of a robot or moves the letters, that robot is discovered and 'dances' into view.

Figure 2. Hurry Screen Shot



```
CLiCK On THe FiSH sO YoU CaN CatCH THem BuT iF YOuR TimE RuNS OuT yOU lOoSe
-ThIS GAMe WaS mAde By BaRNEy nD ADrIaNA
```

Figure 3. Hurry (Loser) Screen Shot



- The player is invested in the outcome, meaning storyline techniques are used to engage the player so he or she is involved in the resolution of the SA program. Ten or 43.5% of the programs use storyline techniques to involve the player in the outcome resolution and, therefore, have narrative tension. For example, the 'Hurry' program (see figure 2) clearly demonstrates an example of narrative tension. The player wants to click on as many fish as possible so that a message stating "WINNER" is displayed when the time runs out. The 'Trouble Maker' program, shown in figure 5, doesn't demonstrate narrative tension. The instructions say, "So who is the trouble maker? If you click on one of the students it will give you the answer!" When the player clicks on a program character, that character runs, or does a cheer, or falls over on the floor. The player may be motivated to click on each character to see what happens with that character; however, there is no time limit or repercussions for not clicking.
- The game has emotional appeal. Because of SA's rich 3-dimensional world of colorful objects and built-in methods, 100% of the programs have emotional appeal. All of the backgrounds and characters that were used in our students' program were 'standard equipment' within SA. In addition to what you see in the figures in this paper, our students used Egyptian, outer space, amusement park, neighborhood, fairy tale, skate park, and boat racing themes.

In summary, 100% of the 23 programs have rules, 56% have goals, 61% have uncertain outcomes, 43% invest the player in the program outcome, and 100% have emotional appeal. According to these defining qualities, many of the students' programs would not be considered games. Given that most students believed they produced games, our quantitative and qualitative results point to the need for work to derive *defining qualities for middle school students' games*.

Figure 4. Find the Robots Screen Shot

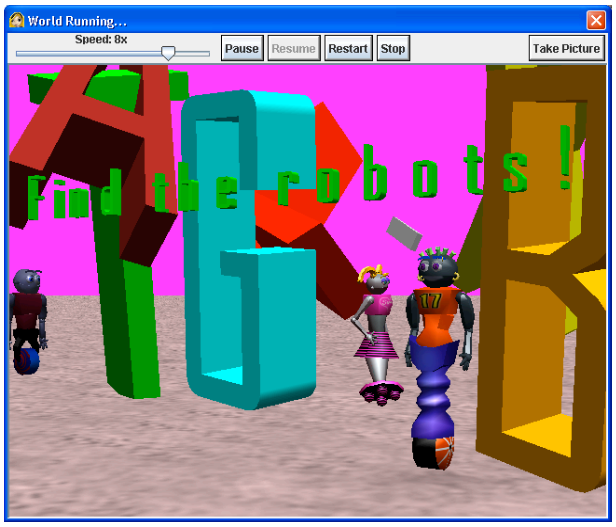
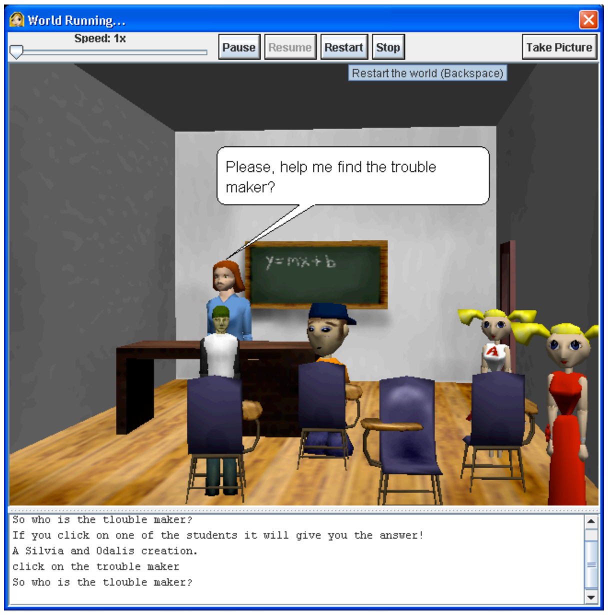


Figure 5. Trouble Maker Screen Shot


World Running...
Speed: 1x
Pause
Resume
Restart
Stop
Take Picture
Restart the world (Backspace)
Please, help me find the trouble maker?
y=mx+b

```
So who is the tlouble maker?
If you click on one of the students it will give you the answer!
A Silvia and Odalis creation.
click on the trouble maker
So who is the tlouble maker?
```

Although the programs created with SA by these middle-school students are not production quality games, as Joel Adams says, "(t)he result is a 3D animated movie (or game) that, while not as polished as a PIXAR production, motivates students by letting them exercise nearly unrestricted creativity" [1]. We found this to be true as well: each program was a unique and creative expression of the students' interests, and all students were extremely proud of their products and the fact that their programs were posted on the Internet. For the purposes of this paper, we consider them games.

From our post-course survey, 90% of the student responders said using the computer was fun and 62% said it is easy to make a game in SA. Some quotes from interviewed students and teachers about the use of SA, making games, and our courses include:

- *"SA is an easy program to use—it has drag and drop instead of coding." (male student)*
- *"(I would use it again) because it's easy, it wasn't overly complicated." (male student)*
- *"(I would use SA again) because it was fun. I might make games my little sister can play and teach her how to make her own games." (female student)*
- *"At break time most of the students wanted to continue instead of taking a break." (teacher)*
- *"I really like using computers a little more and I can make my own games." (female student)*

4. DOES GAME PROGRAMMING WITH SA SUPPORT THE DEVELOPMENT OF IT FLUENCY?

Game programming with SA appeared to have the most potential to build IT fluency in the area of the NRC's fundamental concepts [15], so that is where we focused our initial analyses. We were specifically interested in building IT fluency in the concepts of modeling and abstraction, and algorithmic thinking and programming. Each game was coded for seven aspects of these fundamental concepts. For example, the concept of algorithmic thinking and programming was measured by how students used the five possible ways to control program execution in SA. Games that contain only sequentially executed statements are less sophisticated and can accomplish less than those that contain alternation (if-then-else), iteration (loops), parallelism (do together in SA) and events to control execution. With SA, understanding the concept of modeling and abstraction is shown with the use of programmer-defined methods and by the creation and use of parameters and local and global variables. We analyzed the 23 games and identified those containing student-added instances of these fundamental concepts. Since there are many predefined objects and methods in SA and Alice, we only counted instances of these fundamental concepts when students used them in student-created parts of their games.

The generic SA program consists of a starting object called 'World' and an event that tells SA to send the 'World' object a message to run the starting method called 'my first method.' Therefore, every SA program starts with one event and one object. Additional events can be added and we found that all 23 games have additional student-defined events. Games that contain new methods for built-in objects or new objects show student understanding in modeling and abstraction. For example, a student can combine three existing behaviors of one of their fairy characters (e.g., flap wings, move up, and hum) into a new method called fly. If the student also designs the fly method to be passed a parameter stating the length of time to fly, this represents a more sophisticated understanding (use of parameters and variables) of modeling and abstraction. The percentages of games containing these fundamental concept aspects are: events – 100%, alternation – 26%, iteration – 17%, parallelism – 52%, additional methods – 48%, and parameters, local and global variables – 39%.

The following comments about IT fluency gains are from student post-course interviews and a teacher log.

- *"...(it) feels good to know how to program" (male student)*
- *"(What I like best about SA) is that I get to learn new things like how to make a game and how to program stuff." (female student)*
- *"SA teaches the use of logic and sequencing skills through the development of story and game. Moving from the story level to the game level really helped encourage students to learn the more advanced techniques (variables, when/while, creating new events, etc.)" (teacher)*

We used computer-logging data as part of an effort to understand the process through which programming a game can promote IT fluency. Similar to what Kelleher [14] did, we determined the proportion of instructions of three different types: housekeeping (save and test), programming, and screen layout. For each pair or solo, we identified two dates on which there were sufficient data to analyze: one at the beginning of the course, and one at the end. These data were parsed to reveal frequency of programming, scene layout, and housekeeping instructions. For example, during analysis we found variation in the extent to which students worked on programming compared to working on scene layout. On average, pairs spent more time on programming and housekeeping and solos spent more time on layout. These numbers combine the two dates of logging data for each solo/pair. When we compare the earlier date with the later date, for both pairs and solos, there was an increase in layout tasks and a decrease in programming tasks over time. However, we do not know if that change is due to students doing games as opposed to challenges. In addition, we did not have enough data to investigate whether students who did more programming compared to scene layout had greater increases in IT fluency.

5. DISCUSSION AND FUTURE WORK

Although previous research by Kelleher using a 3-day workshop format found that SA kept middle-school students engaged [15], the benefits of greater exposure were unclear. The results of our pilot study suggest that SA can be used by middle-school students to make games, and that this activity has the potential to promote aspects of IT fluency.

A total of 28 students made 23 games (14 games were made by pairs of students) in just 20 hours even though most of the students did not have programming experience prior to starting the course. Although the vast majority of other classes that teach Alice or SA focus on 3-D stories, we found that SA is very conducive to building programs the students considered games. We used existing literature on game design to identify five aspects of 'gameness:' rules are present, the goal is present and described, the outcome is uncertain, there is narrative tension which invests the player in the game outcome, and there is emotional appeal. In our pilot study, 13 games had at least four of these aspects. These results point to the need for work to derive *defining qualities for middle school students' games*. Feedback on using SA was very positive, students reported low levels of frustration and confusion, and students rated SA as fun and interesting. There were, however, some limitations to the software, such as the number and type of characters and backgrounds, and the difficulty of making some objects move together (e.g., a boy and his skateboard or boats and their passengers). Despite these limitations, the students expressed a great deal of pride in their games and in having their peers and family members play them.

The findings from our pilot study also suggest that creating games with SA does engage middle-school students in some aspects of IT fluency. We coded the 23 games for six different aspects of algorithmic thinking, programming, modeling, and abstraction: events, alternation, iteration, parallelism, methods, and variables/parameters. We found that 30% of the games had four or more aspects and 74% had two or more. A surprising number (52%) included parallelism, a concept that is difficult for novice programmers to learn but is clearly more accessible when using a 3-D animation tool such as SA. Though our sample size was small, the findings from this pilot study suggest that SA can be used by middle-school students with limited computer experience to build games, and that this process can engage students in some critical aspects of IT fluency: algorithmic thinking, programming, modeling, and abstraction. It is just these aspects of IT fluency that are now referred to as computational thinking. Wing states that computational thinking is a fundamental skill for survival in a digital age [25].

We must point out two limitations to these results. First, the high attrition rate limits the generalizability of the findings reported in this paper. Students dropped out for a number of reasons, including the class being too easy or too hard, interest in other activities taking place at the same time (like basketball), and family vacations. As a result, we can say little about what students who dropped out thought about SA. Second, the small sample size means we can say little about the difference between students who worked in a pair, and those who ended up working alone for part of the class. The small sample size also prevented us from looking at differences across demographic groups, and students varied a great deal in English proficiency and access to computers. Finally, we rely on game coding and logging data to determine whether SA engages students in some aspects of IT fluency, but additional individual performance assessments are needed to determine the extent to which students mastered those concepts.

In conclusion, we developed processes and course materials and used them to test the feasibility of Storytelling Alice's use by middle-students to create games. We plan to use these processes and course materials in a larger controlled study of middle-school students using pair programming and Storytelling Alice to create games to determine if this combination can engage youth with little or no programming experience and promote computational thinking.

6. ACKNOWLEDGMENTS

We want to thank the two teachers, Don Jacobs and Shelly Laschkewitsch, for their help. This work was funded by a National Science Foundation grant DRL-0755381. Any opinions, findings, and conclusions or recommendations expressed in the paper are those of the authors and do not necessarily reflect the views of the National Science Foundation.

7. REFERENCES

[1] Adams, J.C. 2007. Alice, Middle Schoolers & The Imaginary Worlds Camps, ACM Special Interest Group on Computer Science Education Conference, (SIGCSE), ACM Press.

[2] Barker, L. and Cohoon, J. 2006. Promising practices: Collaborative learning environments and pair programming. http://www.ncwit.org/pdf/COLLABORATIVE_ED_practice.pdf.

[3] Campe, S., Werner, L., and Denner, J. 2005. Information technology fluency for middle school girls. Published in 8th Annual Computers and Advanced Technology in Education Conference. Internal Association of Science and Technology for Development.

[4] Computer Science Teachers Association. 2003. A model curriculum for K-12 computer science. DOI=http://csta.acm.org/Curriculum/sub/CurrFiles/K-12ModelCurr2ndEd.pdf.

[5] Computer Science Teachers Association Highlighted Resources. Computer Science Teachers Association. Retrieved from http://csta.acm.org/Resources/sub/DownloadableResources.html on December 7, 2008.

[6] Costikyan, G. 1994. I have no words and I must design. Interactive Fantasy #2. Retrieved from http://costik.com/nowords.html on September 2, 2008.

[7] Denner, J. and Werner, L. 2007. Computer programming in middle school: How pairs respond to challenges. Journal of education computing research, 37(2), 131-150.

[8] Denner, J., Werner, L., Bean, S., and Campe, S. 2005. The Girls Creating Games Program: Strategies for engaging middle school girls in information technology. Frontiers: A journal of women studies. Special issue on gender and IT, 26(1), 90-98.

[9] diSessa, A. 2008. Can students re-invent fundamental scientific principles?: Evaluating the promise of new literacies. In T. Willoughby and E. Wood (Eds.), Children's learning in a digital world. Oxford, UK: Blackwell Publishing.

[10] Gee, J.P. 2007. Getting young people to think like game designers. Retrieved from DOI=http://spotlight.macfound.org/main/entry/gee_think_like_game_designers on December 5, 2008.

[11] International Society for Technology in Education 2007. National Educational Technology Standards, 2nd edition. DOI=http://www.iste.org/nets.

[12] Juul, J. 2003. The Game, the Player, the World: Looking for a Heart of Gameness. Level Up: Digital Games Research Conference Proceedings, edited by Marinka Copier and Joost Raessens, 30-45 Utrecht: Utrecht University.

[13] Kafai, Y.B. 2006. Playing and making games for learning: Instructionist and constructionist perspectives for game studies. Game and Culture, 1(1), 36-40.

[14] Kelleher, C. 2006. Motivating Programming: Using storytelling to make computer programming attractive to middle school girls. PhD Dissertation, Carnegie Mellon University, School of Computer Science Technical Report CMU-CS-06-171.

[15] Kelleher, C. and Pausch, R. 2006. Lessons Learned from Designing a Programming System to Support Middle School Girls Creating Animated Stories. In Proceedings of the Visual Languages and Human-Centric Computing (September 04 - 08, 2006). VLHCC. IEEE Computer Society, Washington, DC, 165-172. DOI= http://dx.doi.org/10.1109/VLHCC.2006.30

[16] National Research Council Committee on Information Technology Literacy. 1999. Being fluent with information technology. Washington, D.C.: National Academy Press.

[17] Repenning, A. and Ioannidou, A. 2008. Broadening participation through scalable game design. ACM Special Interest Group on Computer Science Education Conference, (SIGCSE), ACM Press.

[18] Resnick, M., Bruckman, A., and Martin, F. 1996. Pianos not stereos: creating computational construction kits. Interactions 3, 5, 40-50. DOI=http://doi.acm.org/10.1145/234757.234762

[19] Salen, K. 2007. Gaming literacies: A game design study in action. Journal of Educational Multimedia and Hypermedia, 16(3), 301-322.

[20] Shelley, G. B., Cashman, T. J., and Herbert, C. W. 2006 Alice 2.0: Introductory Concepts and Techniques. Course Technology Press.

[21] Werner, L., Campe, S., and Denner, J. 2005. Middle school girls + games programming = information technology fluency. Conference on information technology education. Proceedings of the 6th conference on information technology education. Newark, NJ, USA, 301-305.

[22] Werner, L. and Denner, J. (under review). Pair programming in middle school: Can we describe problem-solving techniques in order to promote them? Journal of Research on Technology in Education.

[23] Werner, L., Denner, J., and Bean, S. 2004. Pair programming strategies for middle school girls. Published in 7th annual computing and advanced technology in education conference. International association of science and technology for development.

[24] Werner, L., Denner, J., and Campe, S. 2006. IT fluency from a project-based program for middle school students. Journal of Computer Science Education Online 2, 205-6.

[25] Wing, J. M. 2006. Computational thinking. Commun. ACM 49, 3 (Mar. 2006), 33-35. DOI= http://doi.acm.org/10.1145/1118178.1118215