Conflictive animations as engaging learning tools

Andrés Moreno **Erkki Sutinen** **Roman Bednarik** **Niko Myller**

Department of Computer Science and Statistics
University of Joensuu,
PO Box 111, FI-80101 Joensuu, Finland
FirstName.LastName@cs.joensuu.fi

Abstract

In this paper we introduce the concept of conflictive animations and discuss its applicability in programming and algorithm design courses. Conflictive animations are animations that deviate from the expected behaviour of the program or algorithm they are supposed to present. With respect to the engagement taxonomy, we propose several ways of learning with conflictive animations. We also initiate a discussion about their possible benefits and drawbacks.

Keywords: Conflictive Animation, Algorithm Animation, Program Visualization, Engagement Taxonomy

1 Introduction

Many teachers have seen how *inadvertently* or *intentionally* being wrong in their lectures has sparked students' attention. In fact, some students will provide feedback only when they can correct the teacher. Moreover, when the teacher introduces errors intentionally, she can assess students' applied knowledge, i.e, whether the students can relate previous information to what is being explained.

Students in a classroom setting may be sufficiently distracted to not absorb what the teacher is saying. This would make it harder for them to later detect teacher's slips when they happen. Algorithm animation tools require the student to be more engaged, and allow for repetition of animations at different speeds according to students' abilities.

Asking programming students to find errors is not new. Rudder et al. (2007) created an online programming course whose materials included a *spot the error* task. Students were asked to identify the logical and syntactical errors in fragments of code, the goal being to evaluate students' comprehension of the basic programming rules.

A different approach is taken in MatrixPro (Karavirta et al. 2004) that addresses the dynamic process of algorithms. MatrixPro is an algorithm simulation tool that contains, amongst others, a *faulty exercise*. In this exercise the student is asked to reproduce the steps that will result in a "inconsistent search tree" according to an incorrect algorithm.

As far as we know, MatrixPro is the only algorithm animation tools that makes use of intentional errors, an incorrect algorithm. However, the possibilities of intentional errors are not fully developed nor considered. MatrixPro designers regard the faulty exercise as a *completely open question* in their taxonomy for algorithm simulation exercices (Korhonen & Malmi 2004).

In this paper we first briefly explain algorithm animation, so as to later explain the concept of *conflictive animation*, and discuss its possible educational benefits. Section 4 suggests possible experimental studies to evaluate the appropriateness of conflictive animations in CS education. We finish the paper with a set of questions that we think are worth considering and the feedback from a survey.

2 Algorithm Animation

Algorithm animation tools visualize the execution of computer algorithms using graphical means. Usually in a step by step manner, animations show how algorithms modify and process given data. Data used in animations consist of characters, numbers, or more complicated data structures such as binary trees or queues.

Teachers and students have used animation tools to teach and study algorithms and programming. For example, sorting algorithms have been taught extensively using animations since Baecker (1981) produced the *Sorting Out Sorting* movie. Visualizations have represented data structures (Karavirta et al. 2004) and virtual machines (Moreno et al. 2004) using a set of graphical primitives: boxes hold values, bars indicate the *weight* of values, arrows serve as pointers, and graphs represent trees or queues.

Hundhausen et al. (2002) summarized the results of several studies on the effectiveness of algorithm visualization. Their meta-study revealed that algorithm animation is educationally effective to a certain extent, in part because of increased student motivation and the time spent with the animations. They emphasized the importance of the way students engage with animations, suggesting that students should regularly work with the animations, rather than just having animations as a uncommon aid.

2.1 Engagement Levels

Recent research (Naps et al. 2002) emphasizes the importance of engagement during interaction with algorithm animation. Naps et al. (2002) explicated several categories of engagement that animation tool creators should consider when designing algorithm animation tools. Their taxonomy lists six levels of engagement, as shown in Table 1.

This taxonomy can be used to help tool designers to add interactive elements to their designs as well as to test hypotheses about the effectiveness of a visualization tool. A way to engage a student with

Level	Well-behaved Animation	Conflictive Animation
No viewing	Tools make no use of animation.	Tools make no use of animation.
Viewing	Students can visualize animations, either step by step or continuously.	Students should be aware of the possibility of viewing an incorrect animation.
Responding	Students are asked to respond to questions related to an animation and the concepts presented in it.	Students have to detect the errors in the animation.
Changing	Students change the animation to explain a different concept that the animation was meant for.	Students change and correct a conflictive animation.
Constructing	Students use the animation tools to create an animation that explains an algorithm or a simpler concept.	Students purposely create a conflictive animation.
Presenting	Students verbally present an animation to an audience.	Students present the conflictive animation and try to confuse peers.

Table 1: Engagement levels and conflictive animations

a visualization is to ask the student to respond to the question 'what happens next in the animation?' (i.e., the program/algorithm execution) (Naps 2005, Myller 2007), or to construct an algorithm visualization using graphical primitives (Karavirta et al. 2004, Rößling & Freisleben 2002).

A complementary way to encourage students to respond to questions during animation would be to ask them to spot errors in it. In this way the question is asked before the animation and they need to pay attention to the animation, a) to understand it and b) to know when an error occurs.

3 Conflictive Animation

Constructivism tenets claim that a student creates knowledge by combining previous knowledge with recent experiences. The resulting mental model may hold misconceptions and not be *viable*. Learning, then, will only happen when students' mental models adapt to the new information and make students reject misconceptions (Smith et al. 1993-1994). The teacher's task is to make students' misconceptions explicit and to guide students towards a viable mental model. Algorithm animations present an opportunity to help teachers and students to identify misconceptions. Conflictive animations are designed to be a tool used by students to reflect on their own mental models. Students should view animations critically, looking for possible errors in them.

We define conflictive animations as those that deliberatedly do not reproduce what the animated code or algorithm is programmed to perform. An animation would advance normally until it starts showing an alternative and potentially wrong execution path. The animation may eventually come back to normality, e.g, showing the correct output. Students should identify when the animation goes wrong, stop it, and reflect on the detected error. The possible errors can affect several levels of the animation, from a simple comparison gone wrong, e.g., evaluating $3 > 4$ as *true*, to higher-level algorithmic concepts, e.g., skipping a balancing step in a red-black tree. An extreme case would be to show an animation of a quicksort algorithm when explaining heapsort.

The idea behind conflictive animations is to add a new dimension to the engagement taxonomy. Conflictive animations could be part of every level in the taxonomy, as shown in Table 1, complementing the educational value of working just with *well-behaved animations*. Students usually have a mental model of how a specific program works before they actually visualize it. Thus, visualizing it should act as a mere proof-check of their beliefs. However, students' knowledge is frequently *fragile* (Oliver & Dalbey 1994), meaning it is incomplete, not recalled, or maybe just wrong. Thus, this would mean that students should benefit from the correct animations: they are complete and correct.

Conflictive animations are graphically described in the Venn diagram shown in Fig. 1. Circle A represents steps in the conflictive animation for an algorithm, and circle B represents steps in which the student spots an error. The intersection of the circles represents the situation in which the student correctly spots the error, a *true positive*. The rest of circle B represents *false positives*, in which the student mistakenly detects an error while the animation is behaving well, and the rest of circle A represents *false negatives*, in which the student fails to detect an erroneous step.

Conflictive animations should try to reveal one or two possible misconceptions that students might have. But students should not only check for the correctness of the visualization. Meaningful conflictive animations should require students to question their own mental models while the animation is running. Students' false positives could reveal further misconceptions beyond those being examined by the animation.

Unfortunately, students commonly use the visualization as just another way to have an algorithm or program executed, focusing only on the final result (Moreno & Joy 2007). This eliminates the potential of animation tools to help students create a viable mental model of the algorithm execution. Applications try to overcome this by increasing the students' engagement level. Some tools ask students to predict the next step in an algorithm, or even to create an animation for themselves.

At times, the answers to the questions posed by the tools can be guessed from cues taken from the last image shown, or from the algorithm pseudo-code. Thus they do not always engage the students in a meaningful learning activity. Furthermore, asking students to produce a new animation may require advanced skills (Rößling & Freisleben 2002) that students would struggle to acquire.

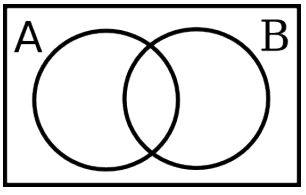

Figure 1: Venn Diagram for Animations

Conflictive animations would require students to follow them carefully step by step. Students should match the graphical representation with the code or concept it actually represents. To be able to spot errors in the animation, students would first have to understand the conventions of the animation, i.e., the meaning of the graphical metaphor it uses to represent the execution. Thus we believe that finding where animations have gone wrong would not only improve students' learning on a particular concept, but give them a better basis for understanding subsequent animations.

3.1 Scenarios

Students should be acquainted with the concepts before testing their knowledge and comprehension with conflictive animations. Their teacher should have already explained the concepts, probably with the use of animations. Then students could view the conflictive animations, knowing that they are conflictive, and try to spot the errors.

Three major roles can be identified in the common scenarios: students, teachers, and peer students. Students could use the conflictive animations to evaluate themselves, or teachers could use them to formally assess students. Peer students could build conflictive animations to challenge other students.

In the first case, the tools should give students instant feedback once they believe they have found the error in the animation. Feedback should at least indicate the correctness of their answer, and a brief explanation of what went wrong in the animation.

The second case could involve written assignments in which students would describe in detail what went wrong in the animation and when. Teachers would be able to detect the common misunderstandings and give personalized feedback to the students.

Finally, conflictive animations could be the basis for a gaming environment. Some students could design and build a conflictive animation, and the other students would try to find the errors in the animation their peers have designed. While building any animation is a complex task, the competitiveness of the assignment may provide additional motivation to students.

3.2 Animations as a story

Interest in CS has seen a drop and student intakes across different CS departments have sharply reduced (Vegso 2005). Alice (Moskal et al. 2004) is an animation tool meant for programming movies and games in an accessible way. Moskal et al. (2004) have shown that Alice appeals to a wide audience and its use increases students' performance and their interest in computer science. Alice's best asset is the ease with which people can use it to *implement* a movie or a game. But another important asset is the playful environment it provides. Students can add a variety of everyday characters and objects and program them to act according to the students' own ideas.

We suggest that algorithm animation tools should add animated characters to animations in order to retain students' interest in the topic, and to attract new students to major in CS. Characters could guide, or misguide, the animation blocks, and describe the animation behaviour. This way, animations will not only consist of boxes, lines and dots moving around, as in Sorting Out Sorting (Baecker 1981), but they will represent a story that can attract a diverse audience.

We envisage an animation story in which two characters collaborate to implement an algorithm. If it is a sorting algorithm, they could move boxes that contain the data to be sorted. However, one character knows the correct steps of the algorithm and the other is wrong. They will discuss and reason about the changes made, e.g., why an array element was selected to be the pivot element. Each will try to convince the other about how to perform the next step. Eventually, the *conflictive character* will manage to convince the *smart character*. This should prompt a reaction in the viewer, who wants the algorithm animation to finish correctly. A set of animations following this idea could be realized to create a multi-episode show to be used in CS lessons at high schools.

3.3 Implementation

Algorithm animation tools such as ANIMAL (Rößling & Freisleben 2002) provide the tools for *visualizers* to create animations of algorithms. We propose that visualizers consider the possibility of creating conflictive animations and uploading them to online repositories[1]. These conflictive animations should be clearly labeled as such to avoid misunderstandings. A start could be made by modifying the scripts of existing animations to introduce errors in them.

JHAVÉ (Naps 2005) could also be extended to support this new way of interaction. Students should be able to push a button to signal that the animation has gone wrong. JHAVÉ would then ask the student at what point the animation went wrong, perhaps using a graphical multiple-choice question that includes snapshots of the animation.

Teachers like using their own examples in their lectures, but they often complain about the time-consuming task of creating animations (Naps et al. 2003). Thus automatic generation of conflictive animations is also important. Teachers could use examples they are comfortable with, and students could test their knowledge with their own programs.

To cater to teachers' needs, generation tools should let the teacher modify certain parameters, e.g., the algorithm to animate, the initial data, the concept to assess, or the level of divergence from correct behaviour. JHAVÉ, for example, already implements the first two parameters. With this information, tools should be able to create an animation plan. Incorrect behaviour can be implemented in the inner algorithms that drive the animations. For example, two different visual routines for animating the balancing of a tree could be implemented, one representing the correct behaviour and the other the conflictive one. The latter would be triggered when the right conditions are met, e.g., a left insertion that prompts a rotation.

Completely automated generation of conflictive programming animations might show to be more difficult to accomplish, as programs created by the students follow a non-deterministic execution behaviour. Algorithms should produce meaningful conflictive animations from students' programs. They should also infer what concept should be conflictive, and when to animate it. A solution for this issue is adaptation, i.e., keeping a user model and a user goal map. Adaptation could keep track of users' current knowledge and learning goals, which vary as the course progresses.

4 Suggested Evaluation

Implementing the concept of conflictive animations is not a trivial task. Existing tools have to be redesigned, both the GUI and the animations they produce. In addition, the new learning task of spotting the errors in animations could confuse the students.

[1] http://www.animal.ahrgr.de/animations.php3

Students should know what the task consists of – detecting an error – and they must be able to solve that task with the tool using the new GUI and animations. Therefore usability evaluation should be carried out from the very beginning in the design phase. Once the task is well defined and the tool is usable, it is time to evaluate the usefulness of conflictive animations.

We propose two main hypotheses to test the possible benefits of conflictive animations:

H1: We hypothesize that students using conflictive animations will understand the concepts explained better than students using a tool at the responding level in the engagement taxonomy, e.g., JHAVÉ. By revealing possible misconceptions, students' knowledge, once fragile, will crystallize. Students will check their assumptions with the animation and discard those that do not correspond to the correct animation.

H2: Students' motivation to view visualizations and animations will increase when they know there are potential mistakes in them.

5 Discussion

We have presented in this paper the concept of conflictive animations, which should reinforce correct mental models in students. The educative effect of intentional errors in instruction is not well researched, and could produce an adverse result. According to our beliefs, conflictive animations should at least increase the attention students devote to animations and their critical thinking.

From our point of view, conflictive animations can at least raise few interesting questions apart from the hypotheses mentioned above:

- Not everybody enjoys being deceived, but does everyone benefit from the deception?
- How can trust in a learning tool be maintained when it is constantly going wrong?

Based on the small-scale survey we carried out at the Koli Calling 2008 conference, CS educators have used some form of conflictive animations or examples and they would be willing to use a tool that provides conflictive animations. We also asked educators to order different scenarios based on their perceived effectivity on learning. We had also classified those scenarios based on the engagement taxonomy classification and interestingly the order of the conflictive animations perceived effectivity does not match the proposed order of the well-behaved animations (Naps et al. 2002). Based on the educators answers, the order of the engagement levels for conflictive animations based on their educational effectiveness from the most effective to the least effective is as follows: changing/correcting, responding/detecting, viewing, constructing, no viewing, and presenting.

References

Baecker, R. (1981), 'Sorting out sorting, color/sound film'.

Hundhausen, C. D., Douglas, S. A. & Stasko, J. T. (2002), 'A meta-study of algorithm visualization effectiveness', *Journal of Visual Languages and Computing* **13**(3), 259–290.

Karavirta, V., Korhonen, A., Malmi, L. & Stalnacke, K. (2004), MatrixPro - a tool for demonstrating data structures and algorithms ex tempore, *in* 'Proceedings of ICALT 2004', pp. 892–893.

Korhonen, A. & Malmi, L. (2004), Taxonomy of visual algorithm simulation exercises, *in* 'Proceedings of the Third Program Visualization Workshop', The University of Warwick, UK, pp. 118–125.

Moreno, A. & Joy, M. (2007), 'Jeliot 3 in a demanding educational setting', *Electronic Notes Theoretical Computer Science* **178**, 51–59.

Moreno, A., Myller, N., Sutinen, E. & Ben-Ari, M. (2004), Visualizing program with Jeliot 3, *in* 'Proceedings of the International Working Conference on Advanced Visual Interfaces, AVI 2004', Gallipoli (Lecce), Italy, pp. 373–380.

Moskal, B., Lurie, D. & Cooper, S. (2004), Evaluating the effectiveness of a new instructional approach, *in* 'SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education', ACM Press, New York, NY, USA, pp. 75–79.

Myller, N. (2007), 'Automatic generation of prediction questions during program visualization', *Electronic Notes Theoretical Computer Science* **178**, 43–49.

Naps, T., Cooper, S., Koldehofe, B., Leska, C., Rößling, G., Dann, W., Korhonen, A., Malmi, L., Rantakokko, J., Ross, R. J., Anderson, J., Fleischer, R., Kuittinen, M. & McNally, M. (2003), Evaluating the educational impact of visualization, *in* 'ITiCSE-WGR '03: Working group reports from ITiCSE on Innovation and technology in computer science education', ACM Press, New York, NY, USA, pp. 124–136.

Naps, T. L. (2005), 'JHAVÉ – addressing the need to support algorithm visualization with tools for active engagement', *IEEE Computer Graphics and Applications* **25**(5), 49–55.

Naps, T. L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S. & Velázquez-Iturbide, J. Á. (2002), Exploring the role of visualization and engagement in computer science education, *in* 'ITiCSE-WGR '02: Working group reports from ITiCSE on Innovation and technology in computer science education', ACM Press, New York, NY, USA, pp. 131–152.

Oliver, S. R. & Dalbey, J. (1994), 'A software development process laboratory for CS1 and CS2', *SIGCSE Bull.* **26**(1), 169–173.

Rößling, G. & Freisleben, B. (2002), 'ANIMAL: A system for supporting multiple roles in algorithm animation', *Journal of Visual Languages and Computing* **13**(3), 341–354.

Rudder, A., Bernard, M. & Mohammed, S. (2007), Teaching programming using visualization, *in* 'Proceedings of the Web Based Education 2007 Conferece (WBE)', ACTA Press.

Smith, J. P., III, diSessa, A. A. & Roschelle, J. (1993-1994), 'Misconceptions reconceived: A constructivist analysis of knowledge in transition', *The Journal of the Learning Sciences* **3**(2), 115–163.

Vegso, J. (2005), 'Interest in CS as a major drops among incoming freshmen', *Computing Research News* **17**(3).