ThinkInk- An Intelligent Sketch Tool for Learning Data Structures

Md Athar Imtiaz
Department of Computer Science
University of Auckland
New Zealand
mimt087@aucklanduni.ac.nz

Andrew Luxton-Reilly
Department of Computer Science
University of Auckland
New Zealand
a.luxton-reilly@auckland.ac.nz

Beryl Plimmer
Department of Computer Science
University of Auckland
New Zealand
b.plimmer@auckland.ac.nz

CHI'18 Extended Abstracts, April 21–26, 2018, Montreal, QC, Canada

ACM ISBN 978-1-4503-5621-3/18/04.
https://doi.org/10.1145/3170427.3188441

Abstract
ThinkInk is an intelligent sketch-based tutoring tool for learning data structures. Our initial evaluation with 45 students shows that they find the tool engaging, fun and a good learning experience. This paper focuses on the interaction design and software engineering required to build such a tool.

Author Keywords
Data structures; tutoring tool; interaction design; intelligent tutoring system; sketch.

ACM Classification Keywords
H.5.2. Information interfaces and presentation: User Interfaces.

Introduction
Data structures, such as arrays, trees and linked lists, are typically visualized as diagrams. These same diagrams are used to explain the operations of algorithms on the data structure. While there have been many WIMP (Windows, Icons, Menus, Pointers) based interactive tools developed to aid students learning data structures – e.g. Vedya [11] and jGRASP [5], these, by their very nature lack the constructionist element of drawing the diagrams and tracing algorithm execution with a pen. Moreover, such tools have been found to have a high cognitive load and distract students from actual learning [6]. Conversely, studies

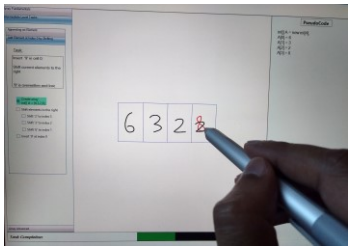

Figure 1a: ThinkInk use

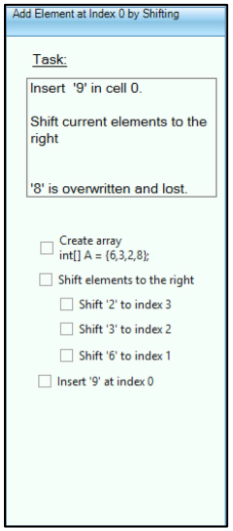

Figure 1b: Task instructions

such as those of Oviatt et al [9] have shown the benefits of pen and paper based sketching for cognition. However, pen and paper cannot provide the real-time feedback of an intelligent tutoring tool to students. A digital ink based sketch enabled tutoring tool can maximize the learning environment by providing both sketch interaction and intelligent tutoring.

The interaction design and software engineering required for such a sketch tool to meet this aim is challenging. In order to provide intelligent support the software must understand the user's sketched input. However sketch recognition is not yet at a level where unconstrained drawing can be reliably recognized. Thus there is a need to carefully design the interaction and the recognizer to maximize both the learning experience and the recognition accuracy. ThinkInk addresses this challenging scenario by taking a task-based approach, which helps both in recognition and learning.

Related work

There have been many data structure tutoring software tools but our search revealed only three sketch-based software tools for learning data structures.

Adamchik [2] reports the first such tool. However it was not evaluated by users. A second sketch-based tool, CSTutor [3] was evaluated by demonstrating the functionality it provided and asking students how useful they thought the tool would be, but the participants in the study never actually used the tool. A prototype of a third tool, CoMo [10], was developed, but it was not evaluated by users.

The presence of just three tools in this domain and lack of credible user evaluation motivated us to fill the gap in this area and hence we have done this study.

ThinkInk

ThinkInk was developed using an iterative design methodology with ongoing informal testing of the user interface and recognizer. The interaction between these two major system components is critically important to the success of the tool. Our approach is to recognize input at each step of the learning task and to visualize the recognizer results at the same time as the tutoring system manages feedback on task completion.

Informal user evaluations of our initial prototypes showed low user approval. This was because of lack of interactive feedback to the user and recognition issues. These issues were rectified by making the tool task-specific, adding intelligent feedback and improving recognition.

The revised tool is designed to be task specific with stepwise feedback. It is also modeless, i.e. the user can sketch and write without having to choose between drawing and writing mode. These features help meet the following important objectives:

1. Improves recognition by limiting the possible user inputs and making them more predictable
2. User interactivity makes the experience engaging and promotes active learning
3. Reduces extraneous cognitive load as it is sketch-based [9]
4. The highly constrained tasks are designed to provide high levels of guidance and feedback

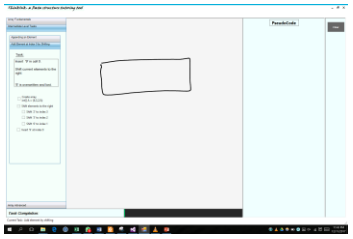

Figure 2a: Rectangle sketch

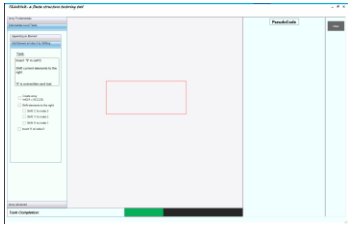

Figure 2b: Rectangle formal

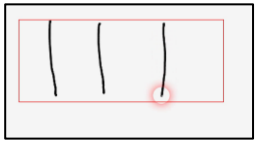

Figure 3a: Splitting into cells

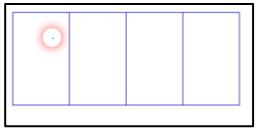

Figure 3b: Vertical lines formal

for students, which is known to be valuable for novices [8].

We followed iterations of paper prototyping, Wizard of OZ, Cognitive Dimensions and informal usability studies, ever mindful of the importance of flow in digital ink environments [7] to reach to the current version of the software. Figure 1a shows the GUI of ThinkInk. It is divided into three main regions. The leftmost area contains the tasks. Next in the centre is the sketching area. In the rightmost panel pseudocode corresponding to a user's sketch is shown.

ThinkInk is implemented using C# Windows Forms and uses machine learning for recognition. RATA [4] is used for shape recognition. Character recognition is achieved using Microsoft Ink DLL. ThinkInks's custom algorithm uses stepwise verification for each stroke in sync RATA and Microsoft Ink. When a stroke is correct and applicable to the step, positive feedback is provided by: formalizing the strokes, printing of pseudocode, highlighting of the correctly performed step, and progression of status bar. If a stroke is not suitable for a particular task step then it is automatically deleted and the user is given hints on what to draw.

Learning Tasks

The current iteration of ThinkInk has 5 tasks for 1-D arrays. Similar tasks are commonly present in data structure course books. These tasks were also informed by the work of Teague and Lister [12], which focused on understanding how novices learn to manipulate arrays. The tasks have been divided into three categories –

- Fundamental level tasks
  - Define and create array
  - Choosing the correct array index
- Intermediate level tasks
  - Appending an element to the end
  - Add element at index 0 by shifting
- Advanced level tasks
  - Sorting using selection sort

The second intermediate level task is used below as an exemplar to illustrate tool use and the various interactions and feedback. The other tasks use similar interactions.

The task is - *shift the elements of array e.g. int [] A = {6,3,2,8} such that all the elements are shifted to the right, 8 is overwritten and lost and the new number '9' gets added in the 0th index location.* Figure 1b shows the task and the instructions. The steps to complete this task are as follows -

Step1: Create an array int [] A = {6,3,2,8}

- The user first sketches a rectangle as shown in Figure 2a. Upon lifting the stylus, feedback is shown by formalizing the rectangle as shown in Figure 2b. The recognition of the rectangle is performed using RATA [4]. If some other figure is drawn, then that sketch is automatically erased and an error message asking the user to draw a rectangle is shown.
- Next, as in Figure 3a, the user draws vertical lines to split the rectangle into cells, which is a common way of representing arrays. RATA is again used for recognition and ThinkInk's algorithm checks if the lines are vertical enough, and whether they are mostly inside

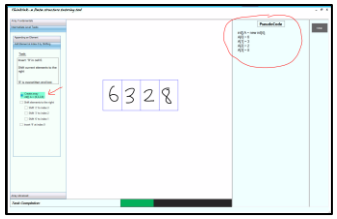

Figure 4a: Pseudocode Feedback

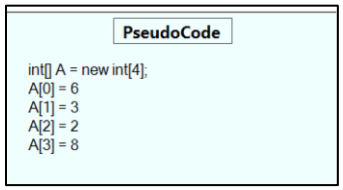

Figure 4b: Pseudocode Feedback

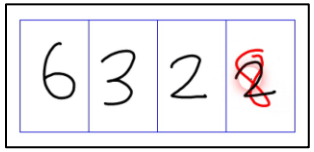

Figure 5a: Shifting 2 to index 3

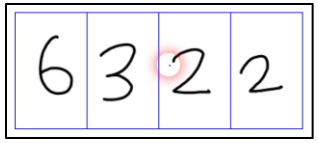

Figure 5b: After shifting 2

the rectangle. If the dividing lines are incorrectly drawn then that ink is automatically erased, and notification to draw vertical lines is shown. The vertical lines are formalized if all conditions are met, as shown in Figure 3b.

- Then the user enters the array elements in their respective cells. Microsoft Ink is used to detect the numbers and ThinkInk's algorithm detects the location of the numbers in the sketching area. If the correct number is entered in the correct cell then pseudocode is generated, as shown in Figure 4a and 4b. If an unexpected element is drawn or an element is drawn in the wrong cell or an elements is drawn outside the cells, it is automatically erased and help messages are provided to guide the user to write the correct element in the appropriate cell.

Step2: Shift elements to the right

- To shift, the user simply writes the element to be shifted in the appropriate cell as shown in Figure 5a. First, the user needs to write 2 to into index location 3 which will result in the overwriting of 8 as shown in Figure 5b. This visualization is made up of two steps – firstly the color of the number to be overwritten is changed to red and secondly there is a pause of 7 seconds after which the number gets deleted and replaced by the newly shifted element. This visualization is akin to the overwriting of an existing element in memory and was particularly liked by the users as they felt that it helped them understand better.
- In case of a wrong number being shifted, it is automatically erased and help notifications are shown to the users.
- The same process is repeated for shifting 3 to index 2 and lastly 6 to index 1

Step3: Insert 9 at index 0

- The user writes 9 at index 0 and once this step is performed correctly a message pops up indicating completion of the task.

Additional feedback is also provided, for example if a step has been performed correctly then it gets highlighted with green, as shown in Figure 6 otherwise, the wrong sketch is deleted automatically and help messages assist the user in performing it correctly, as shown in Figure 7a and 7b. Some steps also have tool-tips that provide more explanation for completing the step.

Evaluation

We evaluated the tool with 45 students. The majority of the participants were novice programmers (30 had programming experience between 1-5 months). Since the target audience of ThinInk are users who have no or little knowledge of data structures, it was ensured that the participants were new to the concepts of 1-D arrays.

Methodology

Participants were given an introduction to the research and a brief demo of the software. Next, they were asked to use the software, which was running on a Microsoft Surface Book with a stylus. After completing the tasks, they answered a 5-point Likert scale

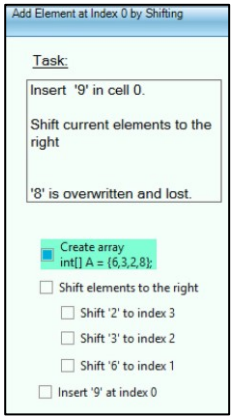

Figure 6: Successful step highlighted

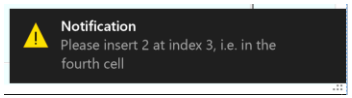

Figure 7a: Help message

Figure 7b: Help message

questionnaire which had questions related to the following constructs derived from Terzis and Economides [13] and Abbad et al [1]–

1. System Interactivity (SIT) –the interactivity and feedback of a system (3 questions).
2. User Interface Design (UID) –the GUI design of a software (6 questions)
3. Perceived Playfulness (PP) –a user's engagement and enjoyment with the software (5 questions)
4. Perceived Ease of Use (PEOU) –the ease of use of using the software(4 questions)
5. Perceived Usefulness (PU) –the usefulness of the software in the user's context (2 questions)
6. Behavioral Intention (BI) –a user's interest in using the software (3 questions).

The average time taken by participants to complete all the tasks at least once was approximately 8 minutes. The majority of the participants performed the Advanced Level task more than once. All the participants successfully completed all the tasks. The overall results indicate a highly positive user feedback. The average user responses for the constructs are shown in Table 1.

In the user study there was no negative feedback for any of the constructs, however, there are a few users who are undecided about the tool. In the case of Behavioral Intention five out of forty five participants are not sure if they intend to use such a software in the future or not. Similarly four users are not very confident about the playfulness or the fun aspect of ThinkInk. Lastly, three users also appear to be undecided about the interactivity and feedback techniques of the software.

Discussion and Conclusions

Our impetus for developing ThinkInk is research indicating the benefits of visualization, interactive feedback and sketching for learning [2; 3; 10]. Moreover, lower cognitive demand of digital-ink and the positive impact of highly constrained tasks on cognition [8; 9] was an added motivator.

While the vast majority of students were positive about all aspects of the tool, a few were neutral. This could be for many reasons: they found the content easy to grasp so did not require any further instruction; insufficient time to get used to the tool; novelty of the interaction with a stylus-based learning tool. No learning strategy is going to be perfect for all students – the sizable positive response suggests our approach works for the majority of students.

We faced many challenges designing and implementing ThinkInk so that it has modeless pen-only interaction which flows seamlessly. Initially problems were caused by minimally constrained tasks with little feedback and low recognition success rates. After adopting a highly constrained task design similar to that used in worked examples [8], through several rounds of design, implementation and user testing the user interaction was constrained and refined. During this process we were cognizant that user flow is crucial for both learning and interaction. Our final prototype provides modeless pen-based interaction where the user can move freely through and between tasks.

	Strongly Agree	Agree	Maybe
BI	16	24	5
PU	29	15	1
PEOU	27	17	1
PP	16	25	4
UID	15	29	1
SIT	17	25	3

Table 1: User opinion. Where the numbers indicate the number of users– BI (Behavioral Intention); PU (Perceived Usefulness); PEOU (Perceived Ease of Use); PP (Perceived Playfulness); UID (User Interface Design); SIT (System Interactivity)

Our tool promotes learning through interactive construction of the data structure visualization and tracing of algorithms. While this study focused on 1-D arrays, we are currently adding binary trees and linked lists to the tool. We also plan to measure the learning gain students achieve with ThinkInk.

References

1 Abbad, M.M., Morris, D., and DE Nahlik, C., 2009. Looking under the bonnet: Factors affecting student adoption of e-learning systems in Jordan. The International Review of Research in Open and Distributed Learning 10, 2.

2 Adamchik, V., 2011. Data structures and algorithms in pen-based computing environments. In Global Engineering Education Conference (EDUCON), 2011 IEEE IEEE, 1211-1214.

3 Buchanan, S., Ochs, B., and Laviola JR, J.J., 2012. CSTutor: a pen-based tutor for data structure visualization. In Proceedings of the 43rd ACM technical symposium on Computer Science Education ACM, 565-570.

4 Chang, S.H.-H., Plimmer, B., and Blagojevic, R., 2010. Rata. ssr: Data mining for pertinent stroke recognizers. In Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium Eurographics Association, 95-102.

5 Cross II, J.H., Hendrix, T.D., Jain, J., and Barowski, L.A., 2007. Dynamic object viewers for data structures. ACM SIGCSE Bulletin 39, 1, 4-8.

6 Dorta, T., 2007. Implementing and assessing the hybrid ideation space: a cognitive artefact for conceptual design. Moon 61, 77.

7 Grosky, W.I., Zeleznik, R., Miller, T., Van Dam, A., LI, C., Tenneson, D., Maloney, C., and Laviola, J.J., 2008. Applications and issues in pen-centric computing. IEEE MultiMedia 15, 4.

8 Kirschner, P.A., Sweller, J., and Clark, R.E., 2006. Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. Educational psychologist 41, 2, 75-86.

9 Oviatt, S., Arthur, A., and Cohen, J., 2006. Quiet interfaces that help students think. In Proceedings of the 19th annual ACM symposium on User interface software and technology ACM, 191-200.

10 Sabisch, A.T.C., 2014. CoMo: a whiteboard that converses about code Massachusetts Institute of Technology.

11 Segura, C., Pita, I., Del Vado Vírseda, R., Saiz, A.I., and Soler, P., 2008. Interactive Learning of Data Structures and Algorithmic Schemes. In International Conference on Computational Science Springer, 800-809.

12 Teague, D. and Lister, R., 2014. Programming: reading, writing and reversing. In Proceedings of the 2014 conference on Innovation & technology in computer science education ACM, 285-290.

13 Terzis, V. and Economides, A.A., 2011. The acceptance and use of computer based assessment. Computers & Education 56, 4, 1032-1044.