TurTan: a Tangible Programming Language for Creative Exploration

Daniel Gallardo, Carles F. Julià and Sergi Jordà
Music Technology Group, Universitat Pompeu Fabra
Ocata 1, 08003 Barcelona
{dgallardo,cfernandez,sjorda}@iua.upf.edu

Abstract

This paper introduces TurTan, a tangible programming language for creative exploration inspired by Logo, which uses a tabletop interface with tangible objects. The aim of this project is to design a toy language for programming entertainment and creative purposes. Along this paper we also discuss some interesting technical issues we have found during its implementation such as tangible linking and angle mapping.

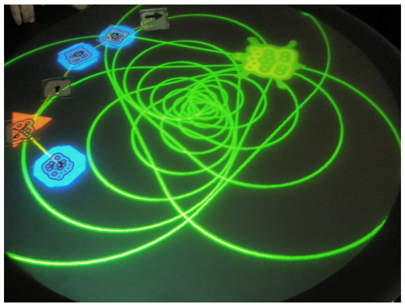


Figure 1. TurTan running

1 Introduction

In the last years many artist-oriented programming languages have arisen. Some of them are still text based, like *processing* or *actionscript*,while others such as *MAX* or *Pure data* use visual paradigms. All of them do still use mouse and keyboard to program, and are more conceived for the creation of interactive art pieces, that for the intrinsic exploration of programming creativity.

Meanwhile, other languages are starting to use tangibles to program. This is the case of the tangible programming systems *Quetzal*[5] and *Bricks*[11]: while the first uses objects to embody the meaning of the instructions (tangibles are just representations), the second embeds electronics on the objects that actually perform the commands themselves. These languages lack however of direct real time control of their parameters, which makes fluid interaction between the programmer and the program difficult to reach.

A more creative-oriented real-time programming language using tangibles is *Task Blocks*[13], which consists of tangible instructions (named task blocks), physical controls and physical connectors.*Task Blocks* is a language for data-processing in which processing modules are programmed connecting task blocks together, while its parameters are controlled by physically wiring the controls to these blocks. This structure allows to have real time control over the program's parameters and structure.

TurTan wants to go one step forward, implementing a toy[10] but fully operational tangible programming language, especially conceived for children and non-programmers, that may help to introduce and explore basic programming concepts in a playful, enjoyable and creative way. It is inspired in Logo[4], and thus designed for performing *turtle geometry*[1]. As the original Logo language, one of the TurTan design goals was to use it for teaching some programming concepts, mainly to children, although no formal tests have been undertaken yet. Unlike the original Logo, TurTan is totally controllable and manipulable in real-time, meaning that there is no distinction between coding and running the program, everything happening simultaneously. In that sense it could also be considered as a *Dynamic programming language*

2 Why tangibles

Taking advantage of the increasingly popular tabletop interfaces and the available fiducial optical tracking systems, TurTan is a programming language that uses physical objects for data manipulation. TurTan's "tangibles" are physical representations of the virtual instructions of a program, which brings a truly experimental approach to programming where anything can be tried and experimented in real-time. It has been frequently reported that the main difficulties that children of 4-7 years deal with when programming, are

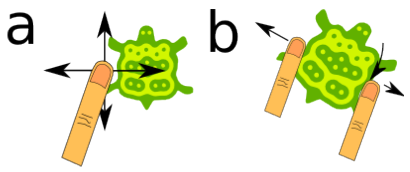


Figure 2. Scale, rotate and translate with fingers

more related with the syntax, with the use of the keyboard and with interpreting error messages, than with the programming concepts themselves[2]. Tangibles have proved on their side to be non intimidating and good promoters of explorative, expressive and experimental activities[12]. In a programming context, tangibles provide an analog-like control over the instructions with immediate real time feedback capabilities, where direct parameter manipulation removes the mediators in the communication process. Furthermore, the social affordances associated with tangibles and tabletops directly encourage concepts such as social interaction and collaboration [6], thus converting programming into a shared and collaborative activity [3].

3 Design and Concepts

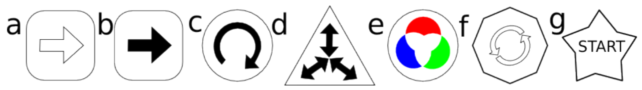
TurTan runs on a tangible tabletop surface, which has been built using a projector and a camera under the table. The camera underneath captures all the activity on the surface, identifying and detecting the positions of the objects and the fingers, while the projector is in charge of the visual feedback of the system. The whole system runs on a single PC as two separated processes: one for the visual tracking, which is done by reacTIVision[9], the open-source tracking software originally designed for the *reactable*[7], and the other for the main TurTan application. The later is in charge of the interaction design, the application logic, and of the visual feedback drawn permanently on the table surface.

TurTan starts with a black screen with the picture of a little turtle in the middle of the surface, and a set of physical objects of different shapes and showing different icons (but all of an average size of about 7x7 cm) randomly disposed around the table. At the beginning, users do not receive any additional information other than the static turtle and the icons displayed on each tangible. Interaction begins when an instruction (a tangible) is placed on the table; some visual feedback is displayed under this block telling the user that the block has been recognized, and the turtle performs the associated instruction accordingly. TurTan generates thus two complementary visual outputs. The first is the virtual program code that is graphically represented around the objects, and the second is the program output. As in Logo, the turtle executes every instruction relative to its own position, instantaneously creating an output vector graphic that is plotted on a virtual canvas, and which is also manipulable through touch gestures. By means of gesture recognition, any user can change the orientation, position or zoom of this canvas, applying one finger for translating (Figure 2-a) and two fingers for scaling and rotating (Figure 2-b), because TurTan interface does not have a preferred orientation. For eliminating both the restrictions of up-down or left-right, and the emergence of leading voices or privileged points-of-view and control which could limit multi-user and collaborative support the table was thought to be circular like the reactable[7].

TurTan can therefore be thought as a dynamic programming language, in which instructions are performed while the user is modifying them, and showing the effects of any instruction in real time, has proved to allow users to quickly understand what they are doing and to easily explore the functionality of each object.

3.1 Instructions

TurTan's programs are composed by a sequence of tangible instructions. Every instruction has one variable parameter which is determined by the rotation the user applies to the corresponding tangible. These instructions do not need to be unique, meaning that several tangibles of the same type can be used in a single program. The instruction types are the following :



Move without painting (a) The turtle moves forward or backward without painting a line.

Move painting (b) The turtle moves forward or backward leaving a coloured line trace.

Rotate (c) The turtle turns right or left.

Scale (d) The turtle scales its body size and also paints its following trace distances at a different scale.

Change colour (e) Changes the line colour.

Repeat (f) The turtle repeats all the actions since the first instruction as many times as indicated.

Start (g) Brings the turtle back to the centre.

4 Tangible Linking

In order to create a tangible programming sequence, users need to establish links between the physical objects and to control the order of these instructions in the more natural and intuitive way. During the project design phase we considered several interaction methods for creating ordered links:

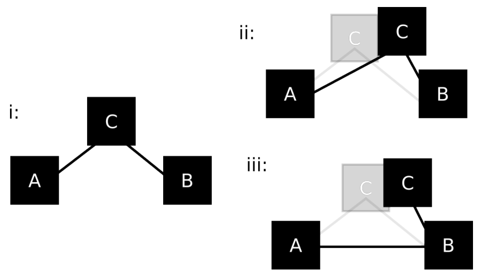


Figure 3. Static (i→ii) and dynamic (i→iii) linking approaches.

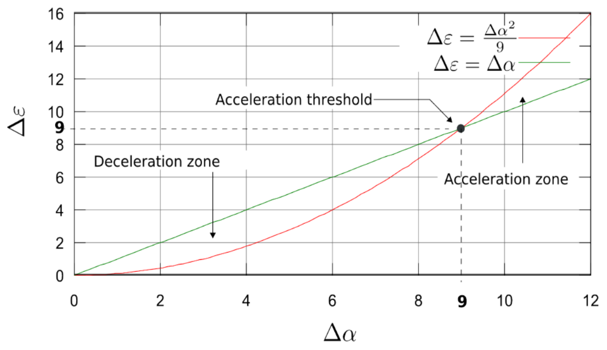


Figure 4. Non-linear scaling of angle.

4.1 Explicit vs. implicit linking

In an **explicit** linking situation, tangible objects are connected by actively defining the link between two or more objects, for example with a manual gesture involving themselves or establishing a physical link using wires. A clear example would be *Quetzal*[5], where users have to link physically the tangibles.

On the contrary, **Implicit links** are established by the system itself with the help of an automatic self-provided algorithm, such as using the relative distance of the neighbouring tangibles or their own angle. An example of this linking process can be extracted from the *reactable*[8], where tangibles are linked between them using proximity rules while the link feedback is given by a virtual wire design representation.

Explicit linking gives the full responsibility of linking tangibles to the user. This makes the act of adding new tangibles a tedious task but gives also a more direct control that avoids computer decisions. Our objective was to find an easy language on which the interaction would be natural and easy. Explicit linking would not only increase the interaction complexity by adding wires or by having to link physically the objects; it would also introduce the idea of wrong behaviour or forbidden gestures, when users would try to make incompatible, syntactically incorrect or meaningless connections. We finally choose implicit linking because of the need to have an intuitive and continuous interaction. Implicit linking requires only the placement of a tangible object on the table in order to link it to a related object within its proximity.

4.2 Dynamic vs. static linking

In implicit linking, more choices have yet to be made. When placing a new tangible instruction on the table, it has to be determined where exactly it will enter into the program sequence by defining its previous and next neighbours. This decision, may depend for example, on a proximity parameter.

By using **static links** (Figure 3 (i→ii)) this decision is applied only once, when the tangible is introduced. Any later modification of the sorting parameter (moving or manipulating a linked object) will not affect its position within the tangible programming sequence. On the contrary, with **dynamic linking**, this order may be modified whenever this sorting parameter is changed, so that moving and manipulating any instruction object on the table might definitely change its position within the program sequence (see Figure 3 (i→iii)).

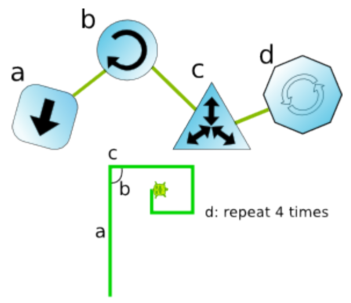
Dynamic linking, which permits to swap positions easily between two elements, is a good choice for applications where the order is not so relevant. Also when there is a large number of objects present on the table, the resulting sequence might be rather complex and the user may have difficulties for anticipating where the tangible will be linked at the moment of introducing it. Dynamic linking therefore allows for easier adjustments of the resulting sequence, while static linking provides a better flexibility for manipulating the tangibles on the table without the danger of destroying the correct order. After some user tests, static linking seemed to work better. Most users thought that the sequence order is essential in a program, and disliked when a single swap would change the nature of the resulting graphical output (Figure 3). However, we also found this swapping behaviour quite intriguing and inspiring, so we have kept both modes for further future user testing.

5 Angle mapping

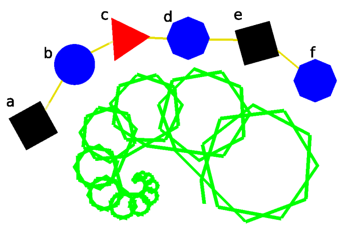
As we mentioned before, the orientation of the objects determines the parameter value of its associated instruction. An interesting issue that we found during the building process of TurTan, was the precision needed for the tangible rotation. When using a linear mapping we had to trade between precision (which implies wasting time spinning the figure to get a higher value) or rudeness (when the angle is multiplied by a higher constant value). In order to solve this issue, we finally applied a nonlinear mapping based on a parabola (Figure 4), which involves the object rotation speed; slower angular speed convey precise movements, while higher angular speed increases the parameter quicker.

6 Program examples

A brief TurTan program example is explained using four tangibles (draw, rotate, scale and repeat). The following figure shows a snapshot of the table surface which combines the instructions (top) and the output results (down). This corresponds to a very simple program as explained with the following pseudocode: *(a)Draw a line, (b)Rotate 90°, (c)Scale 0.75, (d)Repeat.*



The following example uses almost the same instructions, but repeating some of them to create a more complex fractal-like output: *(a)Draw a line, (b) Rotate 120°, (c)Scale 0.9, (d)Repeat, (e)Draw a line, (f)Repeat.*



For more examples you can see some TurTan videos on youtube:

`http://www.youtube.com/watch?v=XK-GNEvQb6Q`
`http://www.youtube.com/watch?v=QQTZsKBMre4`

7 Conclusions and Future work

We oriented TurTan to young children but we still have not made any serious user testing to verify its value as a learning and creative tool. Only several informal tests have been yet undertaken, mainly for identifying conflictive issues in the design and the implementation. We have seen however that TurTan is very easily understood and that users enjoy interacting with it, and we therefore believe that it could constitute a good platform for learning some basic programming language skills.

References

[1] H. Abelson and A. diSessa. *Turtle Geometry, The Computer as a medium for Exploring Mathematics*. The MIT Press, Cambridge, Massachusetts/ London, England, 1980.

[2] Y. E. Alexandrov k., Soprunov S. Logo for the illiterate programmers 20-23. In *Eurologo97*, 1997.

[3] Fernaeus, Ylva and Tholander, Jakob. Finding design qualities in a tangible programming space. In *Proceedings of ACM CHI 2006 Conference on Human Factors in Computing Systems*, volume 1 of *Designing for tangible interactions*, pages 447–456, 2006.

[4] W. Feurzeig, S. Papert, M. Bloom, R. Grant, and C. Solomon. Programming-languages as a conceptual framework for teaching mathematics. In *the National Science Foundation*, 1969.

[5] M. S. Horn and R. J. K. Jacob. Designing tangible programming languages for classroom use. In B. Ullmer and A. Schmidt, editors, *Tangible and Embedded Interaction*, pages 159–162. ACM, 2007.

[6] Hornecker, Eva and Buur, Jacob. Getting a grip on tangible interaction: a framework on physical space and social interaction. In *Proceedings of ACM CHI 2006 Conference on Human Factors in Computing Systems*, volume 1 of *Designing for tangible interactions*, pages 437–446, 2006.

[7] S. Jordà, G. Geiger, M. Alonso, and M. Kaltenbrunner. The reactable: exploring the synergy between live music performance and tabletop tangible interfaces. In B. Ullmer and A. Schmidt, editors, *Tangible and Embedded Interaction*, pages 139–146. ACM, 2007.

[8] S. Jordà, M. Kaltenbrunner, G. Geiger, and R. Bencina. The reactable*. In *Proceedings of the International Computer Music Conference (ICMC 2005)*, Barcelona, Spain, 2005.

[9] M. Kaltenbrunner and R. Bencina. reactivision: a computer-vision framework for table-based tangible interaction. In *TEI '07: Proceedings of the 1st international conference on Tangible and embedded interaction*, pages 69–74, New York, NY, USA, 2007. ACM.

[10] H. F. Ledgard. Ten mini-languages: A study of topical issues in programming languages. *ACM Comput. Surv.*, 3(3):115–146, 1971.

[11] T. S. McNerney. Tangible programming bricks: An approach to making programming accesible to everyone. Master's thesis, MIT, February 2000.

[12] Y. R. Paul Marshall and E. Hornecker. Are tangible interfaces really any better than other kinds of interfaces? In *Pervasive Interaction Labthe*, Open University, Milton Keynes, MK7 6AA, UK, 2007.

[13] M. Terry. Task blocks: tangible interfaces for creative exploration. In *CHI '01: CHI '01 extended abstracts on Human factors in computing systems*, pages 463–464, New York, NY, USA, 2001. ACM.