Is Embodied Interaction Beneficial When Learning Programming?

Pablo Romero[1], Benedict du Boulay[1], Judy Robertson[2], Judith Good[1], and Katherine Howland[1]

[1] University of Sussex, Department of Informatics, Brighton BN1 9QH, UK
[2] Heriot-Watt University, Department of Mathematical and Computer Sciences Edinburgh EH14 4AS, UK
pablor@sussex.ac.uk

Abstract. Embodied interaction has been claimed to offer important advantages for learning programming. However frequently claims have been based on intuitions and work in the area has focused largely around system-building rather than on evaluation and reflection around those claims. Taking into account research in the area as well as in areas such as tangibles, psychology of programming and the learning and teaching of programming, this paper identifies a set of important factors to take into account when analysing the potential of learning environments for programming employing embodied interaction. These factors are formulated as a set of questions that could be asked either when designing or analysing this type of learning environments.

1 Introduction

Often learning environments designed to introduce children to computer programming have used some form of interaction with the physical world. Efforts in this area have tended to concentrate on designing and building environments capable of this interaction but the motivation for the approach as well as the specifics of the instructional design have been driven largely by intuitions. Although there has been some research aimed at building theories and frameworks in areas such as tangibles in learning, computer programming poses specific challenges related to taking advantage of the concreteness of the physical world in order to understand and master an abstract task such as programming.

An embodied type of interaction aims to exploit the familiarity of physical world couplings between actions and their effects by employing analogies based on those couplings [1]. An example of exploiting familiar analogies are electronic organisers that can present documents in portrait or landscape mode depending on how they are physically orientated. Employing analogies of physical world couplings tends to work well for tasks that require a concrete, direct form of manipulation, however computer programming is not about direct manipulation.

Computer programming is related to specifying abstract behaviours to be performed by the computer. These behaviours are abstract because, for instance, they might take place in the future and / or might depend on certain conditions [2]. Programming activities are therefore radically different from direct manipulation tasks

R. Shumaker (Ed.): Virtual and Mixed Reality, LNCS 5622, pp. 97–105, 2009.

and it is not clear whether the benefits of familiar coupling analogies apply in this case. This paper analyses where the benefits of embodied interaction may lie by identifying a set of important factors to take into account when designing or analysing a programming learning environment with embodied interaction. The second section highlights some of the difficulties faced by novice programmers, the third section discusses some of the potential benefits that embodied interaction could offer to the learning of programming, the fourth section describes how embodied elements have been incorporated into learning environments for programming, the fifth presents a set of important factors to consider for this type of environments and the sixth discusses some important aspects of these factors.

2 The Difficulties of Learning Programming

Programming is hard precisely because, among other factors, its abstract nature prevents the use of direct manipulation [2]. Du Boulay [3] separates into five areas the difficulties facing those who are learning to program for the first time. First is a general orientation towards the nature of programming itself. It involves clarifying what programs are for, what can be done with them and what is the point and value of them. Second is the notional machine. This is the abstract machine which will execute the program. We do not mean the hardware or memory registers themselves. It is not about bits and bytes, but about the kind of activities that one can describe in the programming language being learnt. For example, printing a word, causing a Logo turtle to move forward, adding two numbers together, adding a value into a table and so on. The programming paradigm (declarative, functional, object-oriented) is determined by the particular instantiation of the notional machine. Third is the notation of the programming language, in other words the way that the programming language as a language is expressed the syntax, where the semantics is covered by the notional machine. Fourth is standard structures or programming plans [4, 5]. This is about how one puts standard phrases and sentences of the language together to make meaningful paragraphs or essays. For example, how one uses a looping construct to iterate through a list, or how one organises a program into separate methods or separate functions. Fifth is pragmatics: how one makes use of the overall environment (e.g. the editor and the compiler or the program development environment) to get from the idea for a program to a working program itself. Pragmatics also covers developing the skills of effective design and debugging to ensure that the program does what it is supposed to do. Frequently the importance of pragmatics or strategic programming knowledge is underestimated [6].

Typically when learning a second programming language, the general orientation and some aspects of the pragmatics can be generalised from what has already been learned, even if the new notation, the new notional machine and the new structures are quite different. And when learning a third language similarities of notation, notional machine and structures are likely to emerge to simplify and shorted the learning process.

Within a learning environment containing embodied or tangible elements, the question arises as to how far the inclusion of those embodied and/or tangible elements can assist in the mastering of a difficult task such as programming.

3 Potential Benefits of Embodied Interaction

Exploiting the familiarity of physical world couplings between actions and their effects by employing analogies and metaphors based on those couplings is important not only for embodied interaction in general but also for tangibles learning environments. These perceived couplings [7] are an important aspect of the meaningful interaction with the world to which embodied interaction aspires [1]. The couplings can be literal, when there is a close one-to-one mapping in the analogy, or more abstract, when the mapping is looser and the relationship between actions and effects has a certain degree of arbitrariness [8]. Abstract couplings are not necessarily negative, Hornecker and Buur [7] point out that many tangible environments aim for literal couplings missing out on opportunities to exploit people's imagination or to provide useful re-representations of information.

The relationship between actions and effects in abstract couplings is usually mediated by a representation. The more arbitrary the representation the more abstract the coupling. The correspondence could be based on symbols, which have an arbitrary structure, or on icons, which have a more direct perceptual correspondence, for example [9, 10]. Hurtienne and Israel [11] propose that physical manipulations can be employed not only for literal but also for abstract correspondence. They propose to employ the concept of Image Schemas [12], abstract representations of recurring dynamic patterns of bodily interactions, as a sound basis to provide abstract couplings in tangible environments. According to them, Image Schemas capture patterns of sensory-motor experiences, exist beneath conscious awareness and can be represented visual, haptic or kinesthetic way for example. The container schema, for example, is a pattern characterised by comprising an outside, an inside and a boundary between them and is derived from our daily experience with houses, rooms, boxes, cars, etc. Image Schemas can have a central importance in taking advantage of the concreteness of the physical world in order to support the learning of an abstract task such as programming.

Frequently embodied environments mix representations of different types in the perceived couplings, for example, the shape of a toy car could communicate how it could be used but it could also have an attached printed label with symbols to for example, indicate additional functions or characteristics. There are potential advantages and disadvantages in mixing representations of different types; one representation could, for example, constraint the interpretation of another and in this way support the learner [10].

Besides aspects associated with the notion of perceived couplings, embodied interaction can offer other benefits that could be particularly important for programming learning environments. Important aspects in this sense are those that have to do with social interaction and motivation. Embodied interaction has a strong potential for enabling collaboration, which is an important aspect when learning to program [13]. Being able to interact with the environment from multiple points, the inherent visibility of actions and events happening in the physical world and the sheer size of objects and physical environments [7] make embodied interfaces especially suitable for enhancing communication and collaborative learning.

Embodied interaction and its potential for collaboration may also support the understanding of abstraction in a more direct way. Deictic references to physical objects

and gestures performed while communicating have been found to support the emergence of scientific languages and ontologies in school children [14]. Verifying whether this is also the case when using embodied interaction for the learning of programming would be of central importance for the area.

The potential of embodied interaction for motivation was highlighted in one of the first environments employing tangible elements [15]. Embodied interaction was claimed to increase the feelings identification with the characters of the environment and in this way the level of absorption in the task. Additionally, embodied interaction can enable performative action [7], which in turn has been suggested as capable of inducing motivating experiences [16].

4 Embodied Interaction in Programming

The discussion above suggests that there are three important dimensions to consider when employing embodied interaction in learning programming environments: where the embodied element is located (a) pragmatically and (b) conceptually, and (c) what its nature is. The embodied element could target any of the five sources of difficulty for novice programmers outlined in Section 2. For example the embodied element could aim to provide cues about the workings of the notional machine or about the nature of the notation. In practical terms, the embodied element could be associated with any of the elements of the environment: the input, the output, the editor, the debugger, etc. Finally the nature of the interaction could be predominantly haptic or kinesthetic but could combine these with symbolic or iconic elements.

For example, in Logo and its follow-up versions [15, 17–20], the embodied element is associated with their output, a tangible robotic system. Logo aimed to teach programming concepts to children by controlling a robotic turtle. The only tangible element in Logo was its output (the robotic system). The program had to be written by conventional means (typing code to a computer) and the notation was a simplified version of Lisp. Other versions such as the Button Box [18] and Quetzal [20] had additional embodied elements. The button box was a device employed to enable children to control the turtle without having to learn how to type commands on a keyboard. It had a series of buttons that had a one to one mapping with the main controlling functions. Additionally, there was a record button that enabled children to record a sequence of commands and a play button that allowed the command sequence to be played. Although still using a type of keyboard as input, the couplings between actions and effects where more direct than those of a conventional keyboard.

Quetzal is an interesting system that allows children to edit Lego Mind-storms [19] programs with tangible tokens representing keywords of a textual programming language. Children create program statements by physically connecting tokens to form chains that describe the flow of control of the program, similarly to the way textual programs are written as a sequence of statements on the screen with conventional textual languages. In this case there are two independent embodied elements, one is related to the editor of the environment and the other to its output. The editor uses a combination of symbolic and tangible elements.

Another example with a different combination of embodied elements are tangible programming environments. Usually in these systems the focus of the embodied element is the notation. One typical tangible programming environment is the Electronic Blocks system [21]. Electronic Blocks uses Lego blocks augmented with sensors, actuators and logic circuits to enable children to program logical behaviours by joining blocks that perform simple operations. In this case the notation uses a combination of symbolic, iconic and haptic elements. The notation has symbolic elements as blocks of different colours belong to different categories. It also has iconic aspects as the shapes of the blocks indicate their use (for example those shaped as cars can run on wheels). Finally it is haptic as statements of the language are constructed by physically joining the blocks.

The three dimensions discussed in this section plus some of the factors described in previous sections can be used to analyse the learning potential of embodied environments for programming. The following section offers an initial framework that can be used when designing or analysing a programming learning environment with an embodied style of interaction.

5 Important Factors for Embodied Environments for Learning Programming

A set of important factors for programming learning environments with an embodied style of interaction are illustrated on Table 1. These factors could be classified as technical and social. Technical factors could be further classified into those related with the nature of the interaction and those associated with the place where interaction occurs. Social factors could also be further classified into collaborative and those related to motivation. This section talks about them in terms of questions that can be asked when designing or analysing a programming learning environment using embodied interaction.

5.1 Nature of the Interaction

These factors have to do with the type of the perceived couplings, how abstract they are and the type of support they could offer.

- What is the nature of the bodily interaction? It could be symbolic, iconic, haptic, kinetic, gestural, or a combination of them.
- How literal or abstract are the action-effect couplings provided in the environment? If they are abstract are they based on a sound framework of correspondence such as Image Schemas for example?
- What is the support intended through the bodily interaction type? if the interaction type is mixed (iconic and haptic for example), there might be benefits associated with external representations, one representation constraining the interpretation of another for example. This could enable, for example, to allow a progression from understanding more concrete to more abstract notations.

Table 1. Some important factors for programming environments with embodied interaction

Technical	Nature	Interaction type
		Degree of abstraction
		Representational support
	Focus	Programming concepts
		Environment elements
Social	Collaboration	Affordances of embodiment
		Scaffolding abstraction
	Motivation	Possibility of performative action
		Body-syntonic

5.2 Focus of the Interaction

These factors refer to the place where the embodied element occurs. The place could be conceptual (one of the difficult aspects of learning programming) or related to the programming environment (input, output, editor, etc.).

- What programming concept understanding is the bodily interaction aiming to support?
- Do familiar coupling analogies:
- Help to visualise the scope and general orientation of the system?
- Provide cues about the workings of the notional machine and the nature of the notation?
- Constrain or direct users into producing valid structures?
- Offer guidance to perform practical tasks?
- Where in the programming environment is the bodily interaction taking place? It could take place in the input, editor, output, debugger, etc.
- What is the relationship between the targeted concept and the place of the environment where the bodily interaction takes place? For example, is a tangible output aimed to support the understanding of the notional machine?

5.3 Collaboration

One of the most important characteristics of embodied environments is their potential for collaboration. Here we consider, besides the generic collaborative affordances, those that could support the understanding of abstraction.

- What are the collaborative affordances of the bodily interaction style? For example, is the sheer size of the tangible elements conducive to collaboration?
- Are collaborative activities aimed at scaffolding the mastering of abstraction? For example deictic references to physical objects and gestures have been found to support the emergence of scientific language and ontologies in science learning. They might play a similar role in understanding how to specify abstract behaviours.

5.4 Motivation

Similarly to collaboration, motivation is an important factor for embodied environments. For programming, performative action [7] and body-syntonic relations [15] are particularly important.

- Does the system give opportunities for performative action when carrying out the programming task? Combining programming and performative action might induce motivating experiences and appeal to segments of the population who are not usually attracted to programming.
- Is the type of interaction aimed at producing body-syntonic relations with users?

6 Discussion

This paper offers an initial framework that can be useful for analysing the learning potential of programming environments employing an embodied type of interaction. Perhaps more importantly, the initial framework can be used before any system has been built to maximise the learning potential of such environments.

The factors taken into account by the initial framework can be classified into technical and social. Within the technical factors, an important consideration is the degree to which the action-effect couplings provided by the environment are literal or abstract. As mentioned above, embodied interaction aims to exploit the familiarity of physical world couplings but programming, on the other hand, could benefit from employing abstract couplings as a way of specifying abstract behaviours. A concept that can bridge this apparent mismatch is Image Schemas [12]. Image Schemas capture patterns of recurring bodily interactions and therefore encapsulate important aspects of our familiarity with the physical world. At the same time, they are generic enough to be employed for abstract couplings. Image Schemas have been employed to provide abstract couplings in tangible environments [11]. However, they have not, to the best of our knowledge, been employed in programming environments employing an embodied type of interaction. The potential of Image Schemas for this type of environments needs to be evaluated empirically.

Social factors can be particularly important as they can address sociological barriers to programming such as lack of social support and compelling contexts [22]. Social support can be enhanced by the potential of embodied environments for collaboration. The degree of collaboration can be increased if the embodied elements are associated with the actual environment rather than just with its output. Traditional tangible robotic systems, for example, limit the potential for collaboration by employing a conventional desktop setup for the actual programming environment.

Finally performative action can be an important factor for providing compelling contexts. Unfortunately most of the current embodied environments for programming do not provide many opportunities for performative action. These opportunities can be enhanced by environments which embed their embodied elements in large physical spaces (rooms for example) [23] or by those that enable whole body interaction [24].

7 Conclusion

This paper has motivated and presented a set of important factors to take into account when analysing the learning potential of programming environments employing an embodied type of interaction. These factors are classified into technical and social. Technical factors are further classified into those related with the nature of the interaction and those associated with its focus. Social factors have been classified into collaborative and those related with motivation.

The factors are presented as a set of questions that could be asked when designing or analysing programming learning environments employing an embodied type of interaction.

These factors and questions suggest that if the potential of embodied interaction is maximised, the learning of programming would be much more compatible with a studio approach and in many ways similar to learning in disciplines such as architecture or product design. It would be similar not only because the end product could be tangible, but also because of the emphasis on a hands-on approach, on collaboration and on performative action. However this is a conjecture, work that focuses not only on system-building but also on empirically evaluating the benefits and implications of embodied interaction in the learning of programming is needed.

References

1. Dourish, P.: Where the action is: the foundations of embodied interaction. MIT Press, London (2001)
2. Blackwell, A.: What is programming? In: Kuljis, J., Baldwin, L., Scoble, R. (eds.) Proceedings of the 14th annual workshop of the Psychology of Programming Interest Group, pp. 204–218 (2002)
3. du Boulay, B.: Some difficulties of learning to program. In: Soloway, E., Spohrer, J. (eds.) Studying the Novice Programmer, pp. 283–299. Lawrence Erlbaum, Hillsdale (1989)
4. Rist, R.S.: Plans in programming: definition, demonstration and development. In: Soloway, E., Iyengar, S. (eds.) Empirical Studies of Programmers, first workshop, pp. 28–47. Ablex Publishing, Norwood, New Jersey (1986)
5. Gilmore, D.J., Green, T.R.G.: Programming plans and programming expertise. Quarterly Journal of Experimental Psychology 40A, 423–442 (1988)
6. Gilmore, D.J.: Expert programming knowledge: a strategic approach. In: Hoc, J., Green, T.R.G., Samurcay, R., Gilmore, D.J. (eds.) Psychology of Programming, pp. 223–234. Academic Press, London (1990)
7. Hornecker, E., Buur, J.: Getting a grip on tangible interaction: a framework on physical space and social interaction. In: CHI 2006: Proceedings of the SIGCHI conference on Human Factors in computing systems, pp. 437–446. ACM Press, New York (2006)
8. Price, S.: A representation approach to conceptualizing tangible learning environments. In: TEI 2008: Proceedings of the 2nd international conference on Tangible and embedded interaction, pp. 151–158. ACM Press, New York (2008)
9. Purchase, H.: Defining multimedia. IEEE MultiMedia 5, 8–15 (1998)
10. Ainsworth, S.: Deft: A conceptual framework for considering learning with multiple representations. Learning and Instruction 16, 183–198 (2006)

11. Hurtienne, J., Israel, J.H.: Image schemas and their metaphorical extensions: intuitive patterns for tangible interaction. In: TEI 2007: Proceedings of the 1st international conference on Tangible and embedded interaction, pp. 127–134. ACM Press, New York (2007)
12. Johnson, M.: The Body in the Mind: The Bodily Basis of Meaning, Imagination, and Reason. The University of Chicago Press, Chicago (1987)
13. McDowell, C., Werner, L., Bullock, H., Fernald, J.: The impact of pair-programming on student performance, perception and persistence. In: Clarke, L., Dillon, L., Tichy, W. (eds.) Proceedings of the 25th International Conference on Software Engineering, pp. 602–607. IEEE Computer Society, Washington (2003)
14. Roth, W.M., Lawless, D.: Scientific investigations, metaphorical gestures, and the emergence of abstract scientific concepts. Learning and Instruction 12, 285–304 (2002)
15. Papert, S.: Mindstorms: Children, Computers, and Powerful Ideas. Basic Books, New York (1980)
16. Lindley, S.E., Couteur, J.L., Berthouze, N.L.: Stirring up experience through movement in game play: effects on engagement and social behaviour. In: CHI 2008: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems, pp. 511–514. ACM Press, New York (2008)
17. Resnick, M., Martin, F., Sargent, R., Silverman, B.: Programmable bricks: toys to think with. IBM Systems Journal 35, 443–452 (1996)
18. McNerney, T.S.: From turtles to tangible programming bricks: explorations in physical language design. Personal Ubiquitous Computing 8, 326–337 (2004)
19. Boogaarts, M., Daudelin, J.A., Davis, B.L., Kelly, J., Levy, D., Morris, L., Rhodes, F., Rhodes, R., Scholz, M.P., Smith, C.R., Torok, R.: The lego mindstorms nxt idea book: design, invent, and build. Ubiquity 8, 2 (2007)
20. Horn, M.S., Jacob, R.J.K.: Tangible programming in the classroom with tern. In: CHI 2007: extended abstracts on Human factors in computing systems, pp. 1965–1970. ACM Press, New York (2007)
21. Wyeth, P., Purchase, H.C.: Tangible programming elements for young children. In: CHI 2002: extended abstracts on Human factors in computing systems, pp. 774–775. ACM Press, New York (2002)
22. Kelleher, C., Pausch, R.: Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. ACM Computing Surveys 37, 83–137 (2005)
23. Montemayor, J., Druin, A., Chipman, G., Farber, A., Guha, M.L.: Tools for children to create physical interactive storyrooms. Comput. Entertain. 2, 12–12 (2004)
24. Romero, P., Good, J., Robertson, J., du Boulay, B., Reid, H., Howland, K.: Embodied interaction in authoring environments. In: Ramduny-Ellis, D., Hare, J., Gill, S., Dix, A. (eds.) Proceedings of the second Workshop on Physicality, pp. 43–46. UWIC Press, Lancaster (2007)