Towards Collaborative Block-Based Programming on Digital Tabletops

Ben Selwyn-Smith, Michael Homer, and Craig Anslow
School of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
Email: {benjamin.selwyn-smith,mwh,craig}@ecs.vuw.ac.nz

***Abstract*—Block-based programming environments are typically designed for desktop machines or mobile devices. Desktops and mobiles are generally designed for single users to interact with, which makes it hard for multiple users to collaborate effectively. In this paper we explore the possibilities of digital multi-touch tabletops to foster collaborative programming in a block-based paradigm. We note both the different challenges afforded in this new interaction model and the potential benefits unique to collaborative programming with tabletops.**

I. Introduction

Block-based programming languages often have a programming environment where blocks represent program syntax trees as compositions of visual blocks. Digital tabletops are large horizontally-oriented touch-screen devices that many users may work on simultaneously [27]. Block-based languages have shown effectiveness in encouraging engagement, storytelling, and exploratory programming. Digital tabletops foster collaboration between multiple local users, who can work on different tasks in different parts of the table simultaneously, or work side-by-side on the same task together, and split out, come together, or share elements in ad-hoc collaboration.

So far, there is limited work that has combined these two paradigms. Block-based collaboration has largely taken the form of "pair programming" – style sharing of a desktop computer, or networked asynchronous remixing of published projects. Using digital tabletops with block-based languages opens up new collaborative mechanisms, but also requires revisiting some traditional design choices of the paradigm.

In this paper we discuss the advantages – and disadvantages – of such an approach, and the relevant points in the design space of both tabletop interaction and block-based programming. We discuss our preliminary experiences customizing a version of the Tiled Grace block programming environment [14], [15] for multi-user interaction on a tabletop. The next section addresses related work in both block programming and digital tabletops; Section III discusses particularities of the combination of the two paradigms; Section IV addresses various points in the design space we have explored while building our prototype system; and Section V concludes and summarizes our hopes for this novel combination.

II. Related Work

Most block programming systems are designed for use on traditional keyboard-and-mouse environments, constructing programs by dragging blocks with the mouse from some toolbox area onto the workspace. Squeak Etoys [22], Scratch [28], Blockly [7], Pencil Code [3], GP [26], Calico Jigsaw [6], and baseline Tiled Grace [11], [15], [16], [14] all have drag-and-drop as a fundamental element of their interaction model, with drag and target areas suited for traditional WIMP interfaces.

Fig. 1. Two users editing a program in Tabletop Grace simultaneously.

Some block-based languages have targeted touch-screen devices, primarily tablets, including Hopscotch [17], TouchDevelop [18], Blockly [7], and Snap! [10], although for most that was not their primary target. TouchDevelop has touch-screen programming as its main interaction model, providing a point-and-tap structured editor with menus and large touchable areas. Hopscotch allows coding on iPad tablets only and has a similar interaction model to TouchDevelop, but aimed at children.

Tiled Grace [15], [16] is a drag-and-drop block-based programming system for the textual Grace language [4], [5]. The primary distinguishing feature of Tiled Grace is the ability to switch between fully-editable block and textual views of the code, with an animated transition showing the correspondence. Tiled Grace runs in a web browser and uses the traditional toolbox-and-drag approach, similar to the model of Scratch.

Digital tabletops are large touch-screen multi-touch multi-user devices with a horizontal aspect, as contrasted with vertical wall-mounted or free-standing displays [27]. Tabletops allow users to interact with the device from any edge, orientation, and to move freely around the table. Numerous tabletop technologies exist, both commercial and research, but all incorporate some form of touch detection integrated with potentially a high-resolution display.

Tabletop devices have been used for collaboration in many domains, including software development and information visualization among other domains [1]. Isenberg *et al.* [20] highlighted the use of tables in open-ended information foraging tasks, where multiple users could examine, annotate, and share documents with similarity linking to other users, while Kim *et al.* [23] described similar techniques for mixed physical-virtual collaboration between many users. Anslow *et al.* [2] found that tabletops assisted users to work together, break apart, and come back while working on the same broader task of analyzing software through visualizations, but also noted limitations when interacting with distant elements. Bragdon *et al.* [8] adapted the CodeBubbles IDE to touch devices to help support software development team meetings, but used a vertical touch screen. Limited research, however, has focused on collaborative programming with digital tabletops.

III. Facets of Block Programming on Tabletops

Tabletops offer some unique challenges for block-based programming that conventional systems do not address well.

The traditional interaction model of most languages, in the model of Scratch, is to have a toolbox area to one side from which the user can select candidate blocks and drag them into the program. On a tabletop, this interaction breaks down: because the physical size of a table is so much larger, the distance from the toolbox to use may be vast, and could even require moving around the table. If multiple users are editing the same program at once, the toolbox becomes a site of physical contention with users crowding to reach it. Some obvious options here include having toolboxes on each edge, giving each user a personal toolbox / territory they can move with them [29], and allowing a user to call a block into being at any location through a menu or other mechanism.

Drag-and-drop has been seen as a problematic paradigm on traditional keyboard-and-mouse systems, both for block programming [15] and in general [9], [19], [25]. On a mobile touch-screen device, dragging appears more seamless, but the additional occlusion of the finger and hand, and the poor resolution of a fingertip, lead to inaccurate interactions if the touch targets are not large and unambiguous. On a tabletop, however, a drag "across the screen" requires a whole-arm movement or more: for rare interactions, this may be acceptable, but repeated large movements and those requiring prolonged finesse and precision are not viable [30].

Tabletops also exacerbate accessibility problems that most current block-based systems have [24], both of the arm and possibly moving around the edge to reach or see different parts of the program, perhaps particularly in multi-user scenarios as observed by Anslow *et al.* [2]. As well, both the horizontal aspect and the usual height of a table can be significant limitations, as well as the need to see parts of the table clearly at varying distances and viewing angles. Users with impaired vision, stamina, or mobility may find using a tabletop-based block programming system difficult or impossible, perhaps to a greater degree than traditional desktop solutions.

IV. A Tabletop Block Programming System

We have been developing a prototype block programming system for use on a large digital tabletop, building on Tiled Grace with the necessary modifications for the new environment, considering the points noted above. The Tabletop Grace prototype is at the early stages of development and preliminary user studies are under way to measure the perceived effectiveness to support collaborative programming. The system is depicted in use in Figure 1, with two novice programmers engaged in editing at once on the same screen. We are working with a Promethean Activtable (a 16:9 46-inch 32-touch table display), but the experience should generalize to other tabletop systems. The source code of our prototype is available from https://github.com/benmss/Tabletop-Grace.

A. Multi-user Collaboration

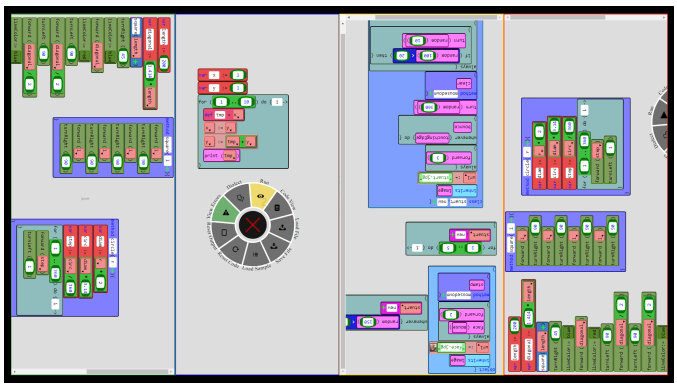
A key goal of using a tabletop is to enable multiple users to work simultaneously, with both in-person and computerized interaction. To avoid contention for the same area, each user can (but is not required to) create their own independent, reorientable, resizable workspace, depicted in Figure 2(a). Users may work on different parts of the program in a separate area of the table (for example, one end each) and then exchange, combine, or share the programs with each other.

Each workspace can be oriented towards any edge of the table through a pie menu obtained by tapping at the edge, and a workspace can be temporarily hidden, moved, or reoriented while retaining its contents. The workspaces are separate modules [12] that can be run and tested individually, but any block or group of blocks can be sent to another workspace through the block's menu, including to a workspace that is currently not displayed, and code reuse (or inheritance [21]) is possible between workspaces. A complete standalone program can be built in parts and assembled in a shared workspace.

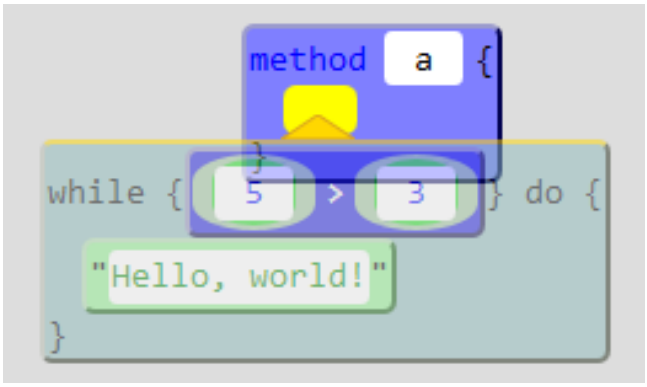
Currently, workspaces have a fixed scale, but are individually scrollable to an infinite area for large programs. Full or partial zooming is a possible future feature, though scaling items to be feasible touch targets is a difficult issue.

B. Floating Pie Menu

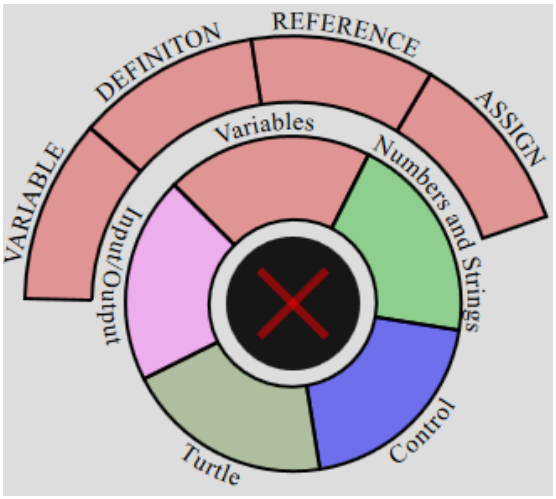
While Tiled Grace relies on the traditional toolbox model, even with potentially-smaller workspaces used by multiple users that model is untenable on a tabletop. Instead, we are allowing tiles to be created in situ by use of a long press and hold which results in displaying a multi-layer pie menu, seen in Figure 2(c). The menu adapts to the set of tiles available in the current dialect [13]. The tile-creation interaction then reduces to primarily a series of taps, while any drag to a



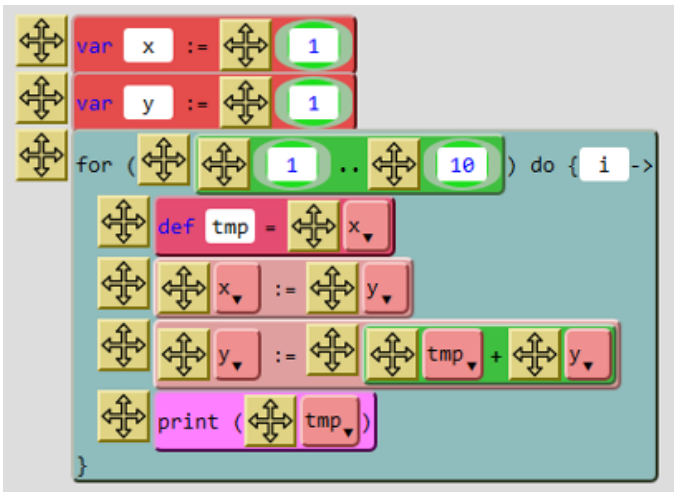
(a) Four workspaces displayed at once in different orientations.



(b) The "pointer" indicating the drop location for a tile being dragged, and the destination hole.



(c) A pie menu partially open to construct a variable-related tile.



(d) The additional "drag widget" in maximal use, which we found impractical.

Fig. 2. Tabletop Grace features.

more specific position is likely to be short. When creating several blocks, simply dragging out of the pie menu rather than tapping creates the tile while leaving the menu open to be used again, giving the affordance of the traditional categorized toolbox with better spatial locality.

Similarly, a long press on a tile creates a pie with deletion, copy, and workspace transfer operations. Both apply to entire nested block structures. A pie for administrative options, such as running, saving, changing dialects, switching to text view, and overlaying errors [15], is available with a two-finger tap and seen in the second (upright) workspace of Figure 2(a).

C. Repositioning Blocks

In the vein of Scratch, Tiled Grace relies on dragging blocks to join them together or separate them. As noted, frequent, fine or lengthy dragging on the table is a problem. Further, the finger and hand used to drag obstructs the user's view of where they are dropping the tile.

We addressed the occlusion problem by adding small "pointer" displays to the top of tiles being dragged, as shown in Figure 2(b). The tip of this triangle indicates the location where the tile will be dropped, rather than the body of the tile itself.

As for dragging in general, the pie menu reduces the incidence of distant movements. Significantly larger tiles are easier to select with a finger, and necessary to be seen clearly across the table in any case. We have experimented with providing an additional fingertip-sized drag "widget" attached to each tile, allowing for even small tiles to be dragged easily. With many tiles nested in close proximity this obstructed the reading of the code more than it helped, which can be seen in the extreme case in Figure 2(d).

We also considered a point-and-click approach: the user would tap on a block once to select it, and then tap anywhere else to move the block to that location. This was the initially-preferred approach, but an ambiguity problem exists: in a shared workspace, a tap from user B should not result in moving a block that user A had selected, but there was no clear way to distinguish this case without adding further steps to the process that reduced the value of the mechanism. As collaborative coding is a key focus of this exploration, we set aside the point-and-tap approach for the moment.

For a single user, however, the technique appears more helpful. In particular, the ability to use both hands to make long-distance moves swiftly by tapping with one hand and then the other in quick succession could be a time-saver and a very natural affordance. We intend to investigate it further, and whether other user-identification techniques could make it a viable model to combine with collaborative block programming. There may also be times when passing a block from user A to user B is in fact the *desired* effect, avoiding the physical coordination of a direct exchange.

D. Text Input

Some parts of programs require text to be written by the user, such as defining the name of a variable or method, or the value of a string to use. While the ActivTable supports a traditional keyboard peripheral, and operating systems built for touch provide soft keyboards as well, the issue of focus is a real one: only one text field can have system-level keyboard focus at any one time, so only one user can enter text at once.

While obvious, this problem was not one we had anticipated until working with the table. Simultaneous text entry is not as rare as expected, with two or three users at once a physical keyboard is hard to manage, and system-level keyboards interact poorly with the browser. Instead, we added separate soft keyboards attached to each text field, with independent text focus. Typing on the touch table remains awkward.

V. Conclusion

Digital tabletops provide a new front for block-based programming and collaboration between multiple users. Tabletops let users work together simultaneously with the ordinary interpersonal affordances, unlike the asynchronous or network-based collaboration that some existing block-based programming systems support. There are also some challenges and limitations associated with using digital tabletops for block programming systems. In this paper we have presented a vision for collaborative block programming with tabletops, highlighting some potential strengths and weaknesses.

Tabletop Grace explores the design space of tabletop block-programming systems. The prototype allows multiple users to work together in their own workspace or otherwise, and modifies existing tabletop interaction methods to suit the block-programming paradigm. We are now in the process of conducting user studies to evaluate the effectiveness of the prototype to support collaborative programming. We hope that further research in this area can open up new and more productive techniques for learners and professional block programmers alike.

References

[1] Craig Anslow, Pedro Campos, and Joaquim Jorge, editors. *Collaboration Meets Interactive Spaces*. Springer, 2016.

[2] Craig Anslow, Stuart Marshall, James Noble, and Robert Biddle. Sourcevis: Collaborative software visualization for co-located environments. In *VISSOFT*. IEEE, 2013.

[3] David Bau, D. Anthony Bau, Mathew Dawson, and C. Sydney Pickens. Pencil Code: Block code for a text world. IDC '15, pages 445–448, New York, NY, USA, 2015. ACM.

[4] Andrew P. Black, Kim B. Bruce, Michael Homer, and James Noble. Grace: The absence of (inessential) difficulty. In *Onward!*, pages 85–98. ACM, 2012.

[5] Andrew P. Black, Kim B. Bruce, Michael Homer, James Noble, Amy Ruskin, and Richard Yannow. Seeking Grace: a new object-oriented language for novices. In *SIGCSE*. ACM, 2013.

[6] Douglas Blank, Jennifer S. Kay, James B. Marshall, Keith O'Hara, and Mark Russo. Calico: A multi-programming-language, multi-context framework designed for computer science education. In *SIGCSE*, 2012.

[7] Blockly Project. Blockly web site. https://code.google.com/p/blockly/.

[8] Andrew Bragdon, Rob DeLine, Ken Hinckley, and Meredith Ringel Morris. Code space: Touch + air gesture hybrid interactions for supporting developer meetings. In *ITS*, pages 212–221, 2011.

[9] Douglas J. Gillan, Kritina Holden, Susan Adam, Marianne Rudisill, and Laura Magee. How does Fitts' law fit pointing and dragging? In *CHI*, pages 227–234. ACM, 1990.

[10] Brian Harvey and Jens Mönig. Bringing "no ceiling" to Scratch: Can one language serve kids and computer scientists? In *Constructionism 2010*.

[11] Michael Homer. *Graceful language extensions and interfaces*. PhD thesis, Victoria University of Wellington, 2014.

[12] Michael Homer, Kim B. Bruce, James Noble, and Andrew P. Black. Modules as gradually-typed objects. In *DYLA*. ACM, 2013.

[13] Michael Homer, Timothy Jones, James Noble, Kim B. Bruce, and Andrew P. Black. Graceful dialects. In *ECOOP 2014*.

[14] Michael Homer and James Noble. A tile-based editor for a textual programming language. In *VISSOFT*. IEEE, 2013.

[15] Michael Homer and James Noble. Combining tiled and textual views of code. In *VISSOFT*. IEEE, 2014.

[16] Michael Homer and James Noble. Lessons in combining block-based and textual programming. *Journal of Visual Languages and Sentient Systems*, 3, 2017.

[17] Hopscotch Technologies, Inc. Hopscotch - learn to code through creative play. https://www.gethopscotch.com/.

[18] R Nigel Horspool, Judith Bishop, Arjmand Samuel, Nikolai Tillmann, Michał Moskal, Jonathan de Halleux, and Manuel Fähndrich. *TouchDevelop: Programming on the Go*. Microsoft Research, 2013.

[19] Kori Inkpen. Drag-and-drop versus point-and-click mouse interaction styles for children. *TOCHI*, 8(1):1–33, March 2001.

[20] P. Isenberg, D. Fisher, S. A. Paul, M. R. Morris, K. Inkpen, and M. Czerwinski. Co-located collaborative visual analytics around a tabletop display. *IEEE Transactions on Visualization and Computer Graphics*, 18(5):689–702, May 2012.

[21] Timothy Jones, Michael Homer, James Noble, and Kim Bruce. Object inheritance without classes. In *ECOOP*, 2016.

[22] Alan Kay. Squeak Etoys authoring & media. Research note, Viewpoints Research Institute, 2005.

[23] KyungTae Kim, Waqas Javed, Cary Williams, Niklas Elmqvist, and Pourang Irani. Hugin: A framework for awareness and coordination in mixed-presence collaborative information visualization. In *ITS*, 2010.

[24] S. Ludi. Towards making block-based programming accessible for blind users. In *Blocks and Beyond Workshop*, pages 67–69. IEEE, 2015.

[25] Scott MacKenzie, Abigail Sellen, and William Buxton. A comparison of input devices in element pointing and dragging tasks. In *CHI*, pages 161–166. ACM, 1991.

[26] Jens Mönig, Yoshiki Ohshima, and John Maloney. Blocks at your fingertips: Blurring the line between blocks and text in GP. In *Blocks and Beyond Workshop*. IEEE, 2015.

[27] Christian Mller-Tomfelde, editor. *Tabletops - Horizontal Interactive Displays*. Springer, 2010.

[28] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. Scratch: programming for all. *CACM*, 52(11):60–67, November 2009.

[29] Stacey D. Scott, M. Sheelagh T. Carpendale, and Kori M. Inkpen. Territoriality in collaborative tabletop workspaces. In *CSCW*, 2004.

[30] Aaron Toney and Bruce H. Thomas. Considering reach in tangible and table top design. In *TABLETOP*, pages 57–58. IEEE, 2006.