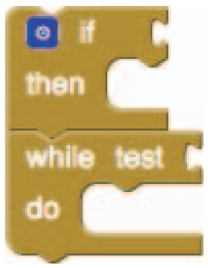
Ten Things We've Learned from Blockly

Neil Fraser
Google
Mountain View, CA, USA
fraser@google.com

***Abstract*—Over the last four years the Blockly team has learned many lessons which are applicable to block-based programming in general. The following are a collection of ten mistakes we have made, or mistakes commonly made by others. Each issue is presented as noncontroversial folk knowledge without supporting data.**

Keywords—block programming; UX; usability; user interface

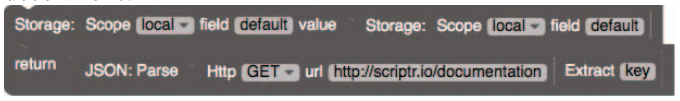
I. Conditionals vs Loops



The most difficult blocks for new users are conditionals and loops. Many block environments add to the difficulty by making both blocks (already similarly shaped) the same colour and placing them in the same category. Users get caught in a spiral of confusion when they accidentally use the wrong block, which adds to the confusion when it behaves contrary to expectation. Blockly moved conditionals into the logic group, changed the colour, and the problem went away.

II. Border Style

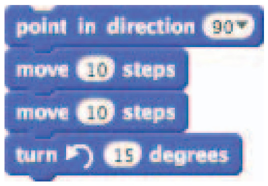
In the 2000s the 'Aqua' look was in style and every onscreen object was decorated with highlighting and shadows. In the 2010s the 'Material Design' look is in style and every onscreen object is simplified to a clean, flat, borderless shape. Most block programming environments have highlighting and shadows around each block, so when today's graphic designers see this they invariably strip off these outdated decorations.


Storage: Scope local field default value
Storage: Scope local field default
return
JSON: Parse
Http GET url http://scriptr.io/documentation
Extract key

As can be seen in the above example of five blocks (from scriptr.io), these 'outdated decorations' are vital for distinguishing connected blocks that are the same colour.

III. Bumping Syntax Errors

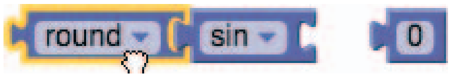
One of the great selling features of block-based programming is the lack of syntax errors. But the screenshot on the right (from Scratch) has an invisible syntax error: the 'turn' block is not connected to the 'move' block above it. There are many ways for this to happen (e.g. place a couple of unconnected 'point' and 'turn' blocks on the workspace, then connect a pair of 'move' blocks to 'point').



The solution is every time a block is moved, look for unattached connector pairs that are ambiguously close to each other and bump one block away from the other.

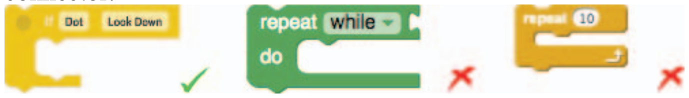
IV. Transitive Connections

Dragging a block should enable that block to connect to others. But what about dragging a block connected to a block and trying to make a connection on the child block? The transitive closure seems like a good idea (which Blockly implemented). But in practice it is a disaster. When working with a large program users will often drag a large assembly partially off-screen to clear room. If transitive connections are allowed, the user will often hear a click sound as the assembly randomly connects to something that was left on the workspace. Transitive connections turn large assemblies into Fluorine atoms they bond with everything.

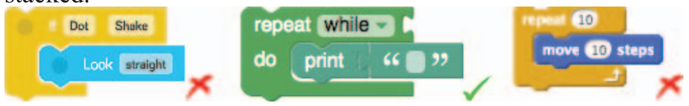


V. Nesting Sub-Stacks

'C' shaped blocks invariably have a connector on the inside-top, but some environments also have a connector on the inside bottom (e.g. Wonder Workshop) whereas others do not (e.g. Blockly and Scratch). Since most statement blocks have both a top and bottom connector, some users do not immediately see that statements will fit inside a 'C' that does not have a bottom connector.



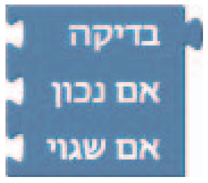
Once users figure out that one statement block fits inside a 'C', they then need to figure out that more that one statement will also fit. Some environments nest the first statement's lower connection into the bottom of the 'C' (e.g. Wonder Workshop and Scratch) whereas others leave a small gap (e.g. Blockly). Snug nesting leaves no hint that more blocks can be stacked.



These two issues interact badly with each other. If an inside bottom connector exists (Wonder Workshop) then the initial statement's connection is made more obvious, but at the expense of the ability to discover stacking. If no inside bottom connector exists (Blockly) then the initial statement's connection is not obvious, but stacking is discoverable. Having no inside bottom connector and nesting the statement's bottom connector (Scratch) appears to combine the worst discoverability attributes of both issues.

Our experience shows that the initial statement's connection is a lesser challenge for users than discovering stacking. And once discovered, the former is never forgotten, whereas the latter needs prompting. Blockly tried both the Wonder Workshop and the Scratch approaches until one day a rendering bug occurred which added the small gap. We saw a marked improvement in user studies due to this bug (now a 'feature' we are proud of).

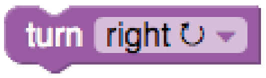
VI. Live Block Images



Documentation for blocks should include images of the blocks it is referring to. Taking screenshots is easy. But if there are 50 such images, and the application is translated into 50 languages, suddenly one is maintaining 2,500 static images. Then the colour scheme changes, and 2,500 images need updating again.

To extract ourselves from this maintenance nightmare, Blockly Games replaced all screenshots with instances of Blockly running in readonly mode. The result looks identical to a picture, but is guaranteed to be up to date. Readonly mode has made internationalization possible.

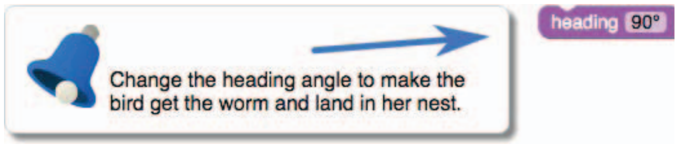
VII. Your Other Left



Feedback from children in the US (though interestingly not from other nations) revealed rampant confusion between left and right. This was resolved with the addition of arrows. If direction is relative (to an avatar, for example) the style of arrow is important. A → straight arrow or a ↱ turn arrow is confusing when the avatar is facing the opposite direction. Most helpful is a ⟳ circular arrow, even in cases where the angle turned is smaller than the arrow indicates.

VIII. Instructions

Blockly Games is specifically designed to be self-teaching, no teacher or lesson plan needed. To accomplish this, the first version of Blockly Games had instructions on each level. Students would not read them. We reduced them to a single sentence, increased the font size, and highlighted them in a yellow bubble. Students would not read them. We created modal popups with the instructions. Students instinctively closed the popups without reading them, then were lost.

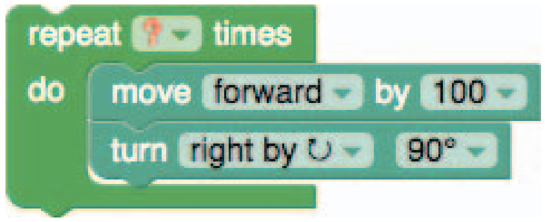


Change the heading angle to make the bird get the worm and land in her nest.

Finally we created popups that cannot be closed. They are programmed to monitor the student's actions and only close themselves when the student has performed the proscribed action. These contextually aware popups are challenging to program, but quite effective.

IX. Code Ownership

Exercises designed to teach a specific concept often provide partial solutions which the student needs to modify to reach the desired effect. A class of non-editable, non-movable, non-deletable blocks was created in Blockly to support this. However, students hated these fill-in-the-blank exercises. They have no sense of ownership over the solution.



Designing free-form exercises that teach the same concepts is more challenging. One technique that has proven successful is to use the student's own solution for one exercise as the starting point for the next exercise.

X. Exit Strategy

Block-based programming is often a starting point for programming. In the context of teaching computer programming, it is a gateway drug that gets students addicted, before moving them on to harder things. Nobody ever got a job because they had three years of Scratch experience. How long this block-based programming period should last for students is hotly debated, but that it is temporary is not debated.

Given this, block-based programming environments used for teaching programming must have an off-ramp appropriate to their students. Blockly Games has four strategies:

1. All text on the blocks (e.g. "if", "while") is lowercase to match text-based programming languages.
2. The JavaScript version of the student's code is always displayed after each level to increase familiarity.
3. In the penultimate game the block text is replaced with actual JavaScript (as shown to the right). At this point the student is programming in JavaScript.
4. In the ultimate game the blocks editor is replaced with a text editor.

Block-based programming environments used for teaching programming need to have a concrete plan for graduating their students. A solid exit strategy also goes a long way towards placating those who argue that block-based programming isn't "real programming".