Taking Stock of Blocks:
Promises and Challenges
of Blocks Programming Languages

Franklyn Turbak
Wellesley College Computer Science Dept.

VL/HCC 2015, Atlanta
October 21, 2015

Talk Road Map
- Motivation: Democratizing Programming
- What are Blocks Programming Languages?
  Demo, History, State of the Art
- Lowering barriers with blocks
  - Syntax
  - Static semantics
  - Dynamic semantics
- Outside the Blocks
- Challenges in blocks programming
  - Usability
  - Learnability in blocks vs. text
  - Perception: blocks programming not “real”, maybe harmful

Alternative Talk Titles that Didn’t Quite Make It

- Why Blocks Programming Matters
- What We Don’t Know About Blocks is a Lot
- Thinking Outside the Blocks

Talk Road Map
- Motivation: Democratizing Programming
- What are Blocks Programming Languages?
  Demo, History, State of the Art
- Lowering barriers with blocks
  - Syntax
  - Static semantics
  - Dynamic semantics
- Outside the Blocks
- Challenges in blocks programming
  - Usability
  - Learnability in blocks vs. text
  - Perception: blocks programming not “real”, maybe harmful
Papert on Constructionism

“The word constructionism is a mnemonic for two aspects of the theory of science education underlying this project … learning is most effective when part of an activity the learner experiences as constructing is a meaningful product.” Constructionism: A New Opportunity for Elementary Science Education (bolding mine)

Maker Movement

“You can innovate as a hobby. Imagine that: a nation of innovation hobbyists working to make their lives more meaningful and the world a better place. Welcome to the maker revolution.” — Mark Hatch, The Maker Movement Manifesto: Rules for Innovation in the New World of Crafters, Hackers, and Tinkerers (bolding mine)

Democratizing Programming

“What we need is a means of democratizing programming, of taking it out of the soulless hands of the programmers and putting it into the hands of a wider range of talents.” Chris Crawford, The Art of Interactive Design

Democratizing Programming

“Digital fluency” should mean designing, creating, and remixing, not just browsing, chatting, and interacting.

Scratch: Programming for All

CACM, Nov. 2009
Democratizing Programming

MIT App Inventor mission statement: The MIT App Inventor project seeks to democratize software development by empowering all people, especially young people, to transition from being consumers of technology to becoming creators of mobile technology.


Target small population
- NYU ITP Teachers on the Run vs. RateMyProfessors.com
- scaling issues unimportant
- simple hardwired data vs. scalable databases
- software for your mom

Leverage small groups
- local knowledge
- trust of other users
- publicly shame deadbeats in group purchase apps

http://shirky.com/writings/herecomeseverybody/situated_software.html

No Texting While Driving App

Daniel Finnegan, English Major, developed the app in Dave Wolber’s USF course CS017: Computing, Mobile Apps, and the Web

App To Track Feral Hogs

Alabama’s Lawrence County High School students used App Inventor to build an app that tracks feral hogs, which were causing economic damage to their community. Their app won a prize of $100K in technology for Samsung’s 2012 Solve for Tomorrow contest.

Trash & Graffiti Cleanup App

East Palo Alto girls created an app to tag the location of trash and create an event for cleaning it up. This app ranked highly in the Technovation Challenge competition.


App to Destroy Mines Safely

Chris Metzger, United States Marine Corps Staff Sergeant, used App Inventor to create an app that helps other Marines destroy weaponry captured in the field. It calculates the amount of explosives necessary to safely destroy captured ammunition and mines.

http://appinventor.mit.edu/explore/stories/united-states-marines-use-app-inventor-field.html

Marriage Proposal App

Hodgson didn't know how to develop an Android app. "How the heck was I going to build this thing?" he recalls thinking. "I tried a couple of other rapid development tools, but they really had too much of a learning curve to let me do it in the timeframe I had in mind." That is, until a friend recommended App Inventor, a tool for amateur Android devs created by Google Labs. "It allowed me, with no java knowledge, to quickly get this thing whipped up," Hodgson says.


Talk Road Map

- Motivation: Democratizing Programming

- What are Blocks Programming Languages?
  Demo, History, State of the Art

- Lowering barriers with blocks
  - Syntax
  - Static semantics
  - Dynamic semantics

- Outside the Blocks

- Challenges in blocks programming
  - Usability
  - Learnability in blocks vs. text
  - Perception: blocks programming not “real”, maybe harmful
Example: Raffle App In App Inventor

http://ai2.appinventor.mit.edu

To enter the raffle, text me now with an empty message: 339-225-0287

Blocks Represent Abstract Syntax Trees (ASTs)

Blocks Languages in the Visual Languages Space

Visual Languages

WIMP Interfaces

DataFlow Languages
( LabView, ProGraph, Show And Tell, DataVis, VPL, VisaVis, …)

Sketch-based, gestural, and tangible user interfaces

Programming By Example

Blocks Programming Languages
( Scratch, Snap!, Blockly, App Inventor, PencilCode, StarLogo TNG/Nova, Alice/Looking Glass, Catrobat/PocketCode, …)

Rewrite Rule Systems
(AgentSheets, Kodu, …)
BLOX (Glinert, 1986)

LogoBlocks (Begel, 1996)

Alice (Pausch et al., 2001)

PicoBlocks (Bonta, Silverman, et al., 2006)
PicoBlocks Passes the “Lucite Test”

PicoBlocks Text/Extension Language

Languages with Physical Blocks

Robot Park (Horn, Solovey, & Jacob, 2007)

Tangible Kindergarten (Bers and Horn, 2009)

Scratch (Resnick et al., 2007)
Scratch (Resnick et al., 2007)

StarLogo TNG (Roque, Wendel, et al., 2007)

- Different plug shapes for different expression types: number, boolean, string, list
- Source of the OpenBlocks Java-based blocks framework

BYOB/Snap! (Harvey, Moenig, et al., starting 2008)

BYOB/Snap! Have First-class Functions
By Feb. 2014:
- 26.5 million participants
- 74% used a blocks language (Code.org Blockly exercises, Scratch, Tynker, Hopscotch, App Inventor, Alice, Looking Glass)
- 17% used a traditional text language (e.g., JavaScript, Python)

As of now: claim 133 million participants

Blocks Languages are Exploding in Popularity!

- 8M registered users
- 11.1M projects shared
- 58M comments posted
- 120K monthly active project creators
- 348K projects by 52K users
- In Sep 2015, 62.7K projects updated by 10.5K distinct users
- BJC used in ~125 schools
Age Distribution: Scratch vs. App Inventor

Talk Road Map
- Motivation: Democratizing Programming
- What are Blocks Programming Languages?
  - Demo, History, State of the Art
- Lowering barriers with blocks
  - Syntax
  - Static semantics
  - Dynamic semantics
- Outside the Blocks
- Challenges in blocks programming
  - Usability
  - Learnability in blocks vs. text
  - Perception: blocks programming not “real”, maybe harmful

Lowering Barriers: Syntax
- Needn’t worry about character-level details and errors (punctuation, capitalization, etc.)
- Recognition from menu of blocks is easier than recall.
- Blocks can have extra annotations to clarify meaning and document sockets.
- Block shapes distinguish expressions, statements, and declarations, preventing fundamental syntax errors.
- Nesting highlights procedure/loop bodies, conditional branches, and scope regions.
- Can move, copy, delete entire syntactically meaningful units.
- Blocks structure emphasizes tree-based nature of programs.

Drop-Downs & Special Editors Reduce Errors & Viscosity
Some Research Questions

1. What are the most beneficial affordances of blocks language syntax? Which of these should be incorporated into IDEs for text-based language?

2. Can we improve line-based debuggers by focusing on syntax nodes instead?

```
var z = g(x) + f(g(y))
```

3. Distinguishing grammatical phrase types:
   - How important is it for blocks representations to distinguish expressions, statements, and declarations?
   - Are distinctions based on plug/socket shape/orientation more effective than others (color, positioning, nesting, etc.)?
   - Are some shapes/orientations of plugs/sockets more effective than others?

Talk Road Map

- Motivation: Democratizing Programming
- What are Blocks Programming Languages? Demo, History, State of the Art
- Lowering barriers with blocks
  - Syntax
  - Static semantics
  - Dynamic semantics
- Outside the Blocks
- Challenges in blocks programming
  - Usability
  - Learnability in blocks vs. text
  - Perception: blocks programming not “real”, maybe harmful
Static Semantics: Name Scoping in AI

- Globals are in a separate namespace
- Indentation visually highlights area of name scope
- Drop-downs list only names in scope.
- Inner names can shadow outer ones
- Changing declared names automatically consistently changes all references

Handling Unbound Names

Static Semantics: What About Types?

App Inventor is dynamically typed, so there's only one plug shape:

Simple “Soft” Static Type Checking in AI

Type errors at block connection time are prohibited by “repulsion”

Dynamic type errors can be hidden by variables:
Connector Shapes in Wellesley PictureBlocks

(Inspired by type-as-shapes in StarLogo TNG)

- **Number**
  - `1`
  - `+`
  - `sqrt`
  - `atan`
  - `x`/`y`

- **Boolean**
  - `true`
  - `not`
  - `and`

- **String**
  - `abc`
  - `num to string`
  - `join`

- **Color**
  - `Red`

- **Picture**
  - `wedge color`
  - `clockwise`
  - `angle`
  - `pic`
  - `Load picname`

- **Polymorphic Sockets**
  - `polymorphic plug`
  - `polymorphic sockets`

- **Polymorphism in PictureBlocks**
  - `choose then`/`else`
  - `abs`

- **Type Blocks**
  - `list of int`
  - `list of (list of string)`
  - `int * string`
  - `bool -> string`

**pushRight: Complete Declaration and Call**

```
pushRight: n p
    return
    test
    then
    empty
    choose
    else
    biside
    pushRight
    get o = 0
    get p
    return
    pushRight
    n = o
    get o = 0
    get p
```

**Type Blocks**

- `List of Integer`
- `List of (List of String)`
- `Integer * String`
- `Boolean -> String`

**Marie Vaske '12 Wellesley**
Type Blocks: More Examples

- listof (string * boolean)
- (listof string) * boolean
- boolean * (string -> listof number)
- (boolean * string) -> (listof number)

Type Blocks: Lists

Type Blocks: ML Style Universal Polymorphism

Some Research Questions

1. Are plug/sockets shapes that distinguish expression types (e.g., in StartLogo TNG, PictureBlocks) beneficial?

2. How understandable are various approaches to expressing sophisticated types and polymorphism visually?

3. For statically typed language L (e.g. C, Java, ML, Haskell), can we design a blocks system for L that accurately and understandably represents types visually?
Talk Road Map
- Motivation: Democratizing Programming
- What are Blocks Programming Languages?
  Demo, History, State of the Art
- Lowering barriers with blocks
  - Syntax
  - Static semantics
  - Dynamic semantics
- Outside the Blocks
- Challenges in blocks programming
  - Usability
  - Learnability in blocks vs. text
  - Perception: blocks programming not “real”, maybe harmful

Stepping in PencilCode, early Scratch

Variable Display in Scratch

App Inventor: DoIt
Simple form of interactivity/liveness found in many blocks environments (as well as interpreter text-based languages).
Better Debugging: Watch

Johanna Okerlund ’14
Wellesley

Emery Gerndt Otopalik ’16
Wellesley

Some Research Questions

1. What are better ways to show the execution of blocks programs? How to handle the visualization of function calls and complex data?

2. What information is most useful for helping blocks programmers debug typical errors?

3. What to do in cases (robotics, mobile apps) in which the program is running on a device other than the computer in which it’s been written?

Talk Road Map

- Motivation: Democratizing Programming
- What are Blocks Programming Languages? Demo, History, State of the Art
- Lowering barriers with blocks
  - Syntax
  - Static semantics
  - Dynamic semantics
- Outside the Blocks
- Challenges in blocks programming
  - Usability
  - Learnability in blocks vs. text
  - Perception: blocks programming not “real”, maybe harmful

Thinking Outside the Blocks: Abstraction
Thinking Outside the Blocks: Abstraction

What does this code do?

```
when TextSMS.MessageReceived
  number = messageText
  set TextSMS.PhoneNumber to get number
  call TextSMS.SendMessage
  call TextToSpeech1.Speak
  message = join ["New text from ", get number, ", The message says ", get messageText]
```

Thinking Outside the Blocks: Abstraction

```
onEvent {v"myCanvas", v"mousedown", function (event)
  setFillColor ("blue")
  circle [event.clientX, event.clientY, 10];
};
onEvent {v"clearButton", v"click", function (event)
  clearCanvas();
};
```

Thinking Outside the Blocks: Community

```
when Canvas.TouchDown
  do call Canvas.Clear
```

Thinking Outside the Blocks: Browser-Based Environments & Cloud Program Storage
Some Research Questions

1. Which programming abstractions developed in blocks languages are worthwhile to incorporate into traditional languages.

2. Can we learn something from sharing/remixing communities in blocks languages that we’re not learning from communities (e.g. forums) for other languages?

3. What kinds of analysis can be done on the massive cloud data collected for user blocks programs to better understand their learning, help them to debug their programs, and improve the programming environments they use?

Talk Road Map

- Motivation: Democratizing Programming
- What are Blocks Programming Languages?
  - Demo, History, State of the Art
- Lowering barriers with blocks
  - Syntax
  - Static semantics
  - Dynamic semantics
- Outside the Blocks
- Challenges in blocks programming
  - Usability
  - Learnability in blocks vs. text
  - Perception: blocks programming not “real”, maybe harmful
Usability: Organizing 2D Blocks Workspaces

Shirley X. Lu ’15  
Wellesley
Devid Farinelli ’16  
U. of Bologna

Folders in App Inventor (under development)

Usability: Reusing & Sharing Blocks Programs

Backpack in Scratch and App Inventor

Usability: Droplet’s Isomorphic Blocks/Text Conversion

Used in PencilCode and Code.org’s AppLab JavaScript curriculum

AI: Conversion Between Blocks and Text

Karishma Chadha ’14  
Wellesley
Some Research Questions

1. Which aspects of 2D blocks layout are beneficial and which are not?
2. What are effective ways to create, organize, search, and navigate 2D blocks programs? Can we do these on small screens?
3. What are effective ways to leverage the best aspects of blocks and text when creating and manipulating programs?
4. What tools can better support the collaborative construction of blocks programs? (Neil Fraser and Mark Friedman at Google have done preliminary work on a kind of Google Docs for Blocks.)
5. How can we improve accessibility of blocks programming environments for the blind and visually impaired?

Learnability in Blocks vs. Text

- Lewis: Logo vs. Scratch study (SIGCSE 2010)
  - Few significant differences between Logo-first and Scratch-first
  - Scratch-first did better on conditionals
  - Logo-first had more confidence as programmers.
- Meerbaum-Salant, Armoni, & Ben-Ari (ITiCSE 2011)
  - Scratch programmers have undesirable “habits of programming”: bottom-up programming & extremely fine-grained programming
- Weintrop and Wilensky: Snap! vs Java (IDC 2015)
  - Blocks easier to read and compose than text
  - Blocks perceived as more verbose, less powerful, less authentic
- Problem: Nonisomorphic languages
  - Weintrop and Wilensky Commutative Assessment on blocks vs. text in isomorphic languages (ICER 2015) is promising approach
  - Matsuzaka taught Java with blocks environment isomorphic to text (SIGCSE 2015). Students perceived text as more “real”.

Talk Road Map

- Motivation: Democratizing Programming
- What are Blocks Programming Languages? Demo, History, State of the Art
- Lowering barriers with blocks
  - Syntax
  - Static semantics
  - Dynamic semantics
- Outside the Blocks
- Challenges in blocks programming
  - Usability
  - Learnability in blocks vs. text
  - Perception: blocks programming not “real”, maybe harmful

Some Research Questions

1. Do Matsuzaka’s results hold in other examples in which blocks and text language are very similar or isomorphic? Tools like Droplet should make this easier.
2. If people transition from a blocks language to a more traditional text language, what concepts and skills are transferred? What difficulties are encountered? What kinds of explicit instruction can aid this transfer?
3. Are there particular transition paths from blocks languages to traditional languages that tend to have better concept and skill transfer than others?
Talk Road Map

- Motivation: Democratizing Programming
- What are Blocks Programming Languages?
  - Demo, History, State of the Art
- Lowering barriers with blocks
  - Syntax
  - Static semantics
  - Dynamic semantics
- Outside the Blocks
- Challenges in blocks programming
  - Usability
  - Learnability in blocks vs. text
  - Perception: blocks programming not “real”, maybe harmful

Negative Responses to Blocks Languages

I have never met a student who cut their teeth in any of these languages and did not come away profoundly damaged and unable to cope.

I mean this reads to me very similarly to teaching someone to be a carpenter by starting them off with plastic toy tools and telling them to go sculpt sand on the beach.

Not one thing they learn will bear any piece of resemblance to real work. All you’re doing is teaching them misimpressions of what the job is, and tricking them out of having meaningful formative experiences.

http://blog.acthompson.net/2012/12/programming-with-blocks.html

Working with actual code writing instead of a drag & drop interface prepares children better for the real world.

http://www.playcodemonkey.com/

Mark Sherman’s Response

So they currently see this:

when it is really this:

Why do they see it this way? Because they grew up on this:

Yes, it is colorful and newfangled, but it still gets jobs done. Not all of them, but a bunch of them.

More Positive Feedback

I would like to express my utmost appreciation for your product. I’m teaching several pre-CS courses for gifted youth at Junior-high school level (7th-9th grades) as well as CS and software engineering at high school (10th – 12th grades) including Android development in Java. It is really amazing that in AppInventor, 7th grade students (with about 50 hours prior experience in Scratch) can do in 6 hours what 12th grade students take about 200-300 hours to achieve in Java (and this is after studying CS and Android development for about 700 hours). AppInventor goes way beyond the 80:20 principle (80% of the utility in 20% of the effort) – it is more like 60:5 (60% of the functionality, for less than 5% of the effort) which makes it much more fun, and opens up a lot of space for creativity.

Yossi Yaron, Israeli teacher
Some Research Questions

1. In what substantive ways are particular blocks programming languages inferior to text languages with which they are being compared?
2. What can be done to make blocks languages appear more “real” to novice programmers?
Questions?

Research Questions are at http://tinyurl.com/VLHCC15Blocks