Creating Data-driven Feedback for Novices in Goal-driven Programming Projects

Thomas Price
Tiffany Barnes, Advisor

North Carolina State University

AIED Doctoral Consortium 2015

Motivation

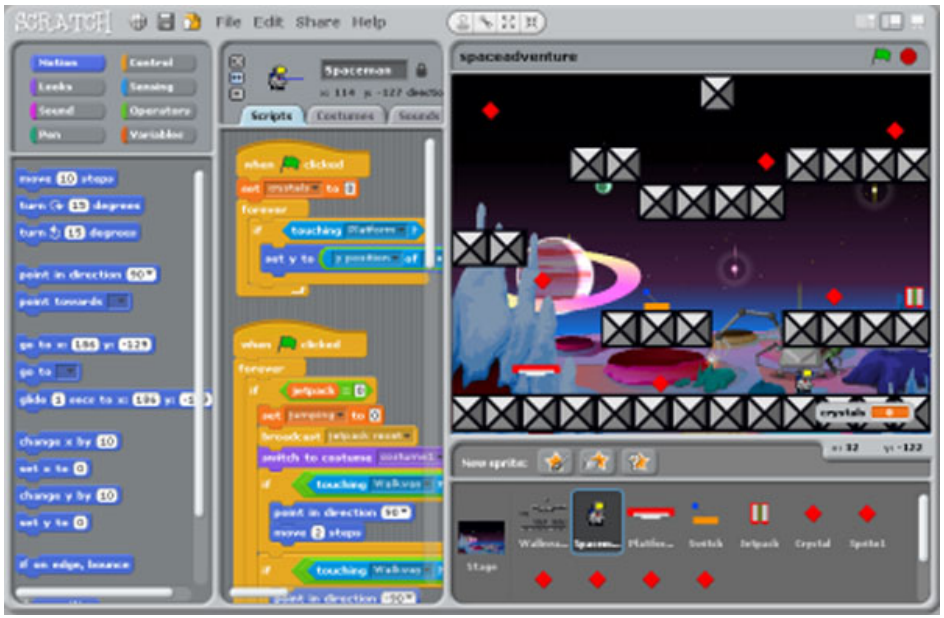
Goal

To provide ITS-style feedback, in the form of on-demand hints, to students working in media-rich, novice programming environments

- Examples include Scratch (Resnick et al. 2009), Alice (Cooper, Dann, and Pausch 2003) and MIT AppInventor (Pokress and Veiga 2013)
- Allow users connect with their interests: games, apps, stories
- Can improve grades, comprehension and interest in computing (Moskal, Lurie, and Cooper 2004; Meerbaum-Salant, Armoni, and Ben-Ari 2010)
- Without instructors, they lack support for struggling students
  - In after-school programs (Maloney et al. 2008) or at home

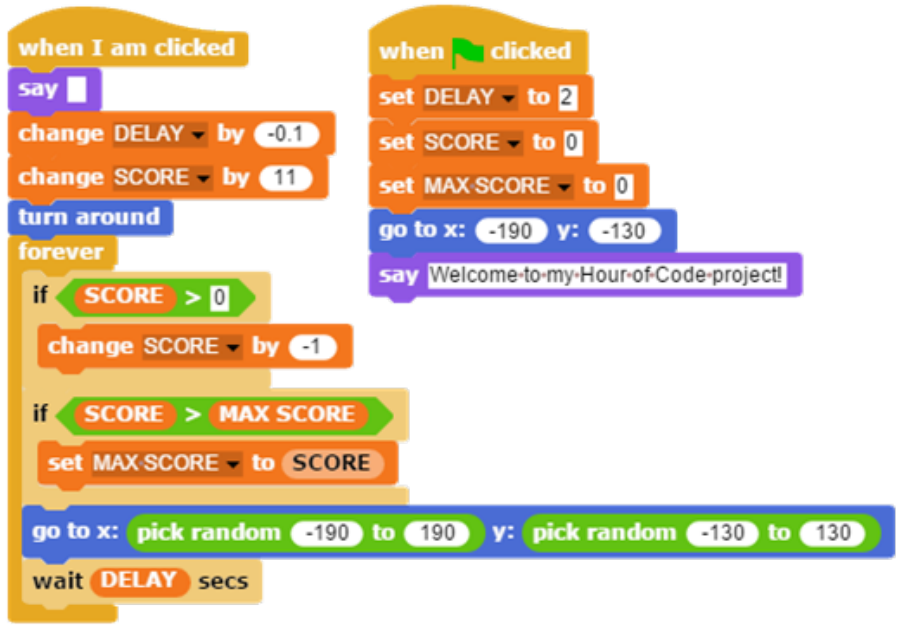
Goal-driven Programming Projects

- Address some open-ended, creative design task
- Comprised of multiple, interdependent subgoals
- Output is unstructured and success may be subjective
- Makes the projects interesting, but also makes hints difficult to generate



Goal-driven Programming Projects

- Address some open-ended, creative design task
- Comprised of multiple, interdependent subgoals
- Output is unstructured and success may be subjective
- Makes the projects interesting, but also makes hints difficult to generate



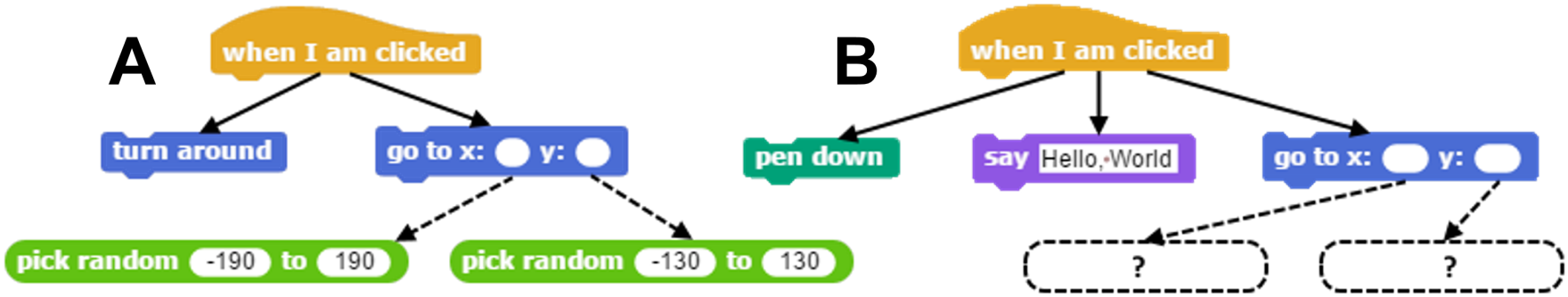
Current Approaches to Hint Generation

- Hints can be hand-authored or generated by an expert model, but this can be time-consuming and inflexible
- The Hint Factory is a data-driven approach that uses past students' solutions to generate hints (Stamper et al. 2013)
- We match a student's current state to other students who have solved the problem, and generate advice based on their solutions
- Has been adapted to work with simple programming problems, using techniques to increase state overlap (Rivers and Koedinger 2013; Jin, Barnes, and Stamper 2012; Hicks, Peddycord III, and Barnes 2014)
- Goal-driven projects produce especially sparse state-spaces, so how do we match students to past data?

The Subtree Approach

- Represent a student's code as an abstract syntax tree (AST)
- Isolate the subtree which encompasses the student's most recent actions
- Search for matching subtrees in previously observed states on known solution paths
- Advise the student to modify their subtree as the successful student did
- **Advantage**: greatly increases the probability of matching a previous student
- **Limitation**: ignores possibly relevant code outside the subtree, lowers hint quality

An Example



- Say we have previously observed Student A (left)
- We can hint Student B (right), even though the rest of the program is different
- We use `[go to x y]` as the root of the subtree

Current Progress

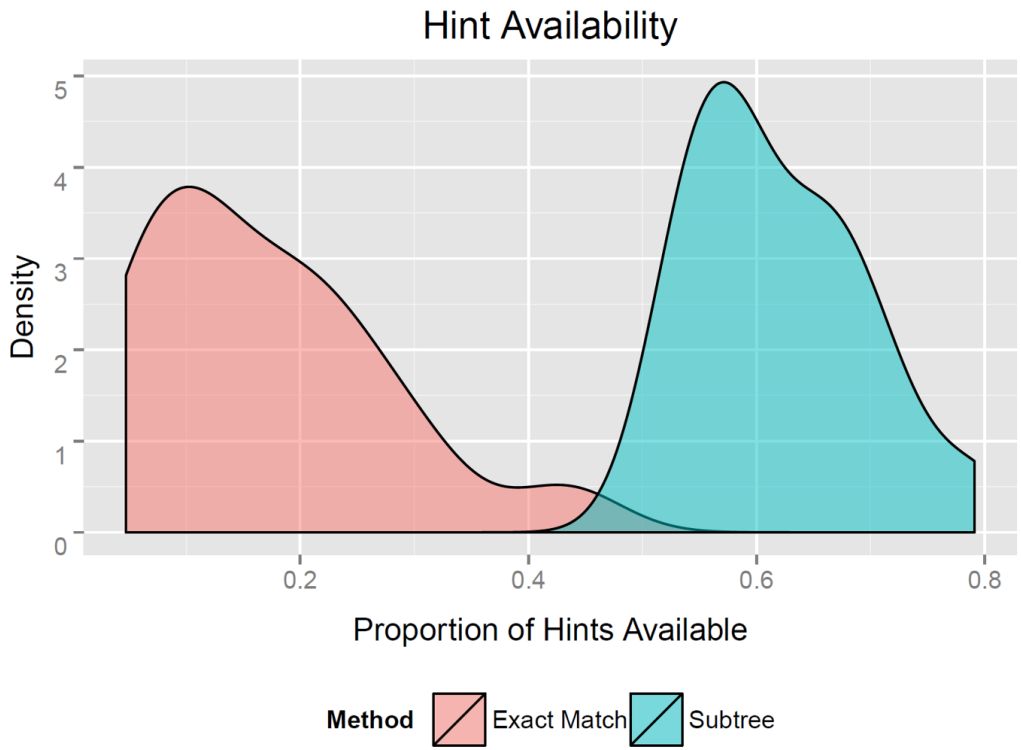
- Collected a dataset from 17 middle school novices working on a goal-driven program during a STEM outreach program
- Student solutions were too diverse to apply current methods only, e.g. canonicalization (Rivers and Koedinger 2012):

	Raw	Canon.	Ordered
Total States	2380	1781	1656
Percent Unique	97.5%	94.8%	92.8%
Mean Non-Unique Freq.	3.44	3.95	2.82
Mean % Path Unique Standard Deviation	89.9% (6.67)	83.0% (10.5)	78.9% (13.3)

- Caveat: this is a *very small* dataset

Some (*Very*) Preliminary Results

- With exact (canonicalized) state matching, only an average 17% of the states in a student's path matched another student
- Using subtree matching, this rose to 62%



Future Work

- Collect a larger corpus of student solutions
- Refine the subtree approach to balance hint quantity and quality
- Generate a set of hints and integrate them into the programming environment
- Collect a second dataset from students who have on-demand hints available
- Compare students' performance and self-efficacy with and without data-driven hints

Feedback Wanted

- What should the balance be between hint quality and quantity?
  - How confident does a tutor need to be before offering advice?
- What programming concepts should be included in the goal-driven programming activity?
  - How complex should it be?
- Are there additional performance measures I should consider collecting?

Thank you!

Questions? Comments?

References I

Cooper, S, W Dann, and R Pausch (2003). "Teaching objects-first in introductory computer science". In: *ACM SIGCSE Bulletin*.

Hicks, A, B Peddycord III, and T Barnes (2014). "Building Games to Learn from Their Players: Generating Hints in a Serious Game". In: *Intelligent Tutoring Systems (ITS)*, pp. 312–317.

Jin, W, T Barnes, and J Stamper (2012). "Program representation for automatic hint generation for a data-driven novice programming tutor". In: *Intelligent Tutoring Systems (ITS)*.

Maloney, John et al. (2008). "Programming by choice: urban youth learning programming with scratch". In: *ACM SIGCSE Bulletin* 40.1, pp. 367–371.

References II

Meerbaum-Salant, Orni, Michal Armoni, and Mordechai Ben-Ari (2010). "Learning computer science concepts with scratch". In: *International Computing Education Research Conference 2010 (ICER '10)*, pp. 69–76.

Moskal, B, D Lurie, and S Cooper (2004). "Evaluating the effectiveness of a new instructional approach". In: *ACM SIGCSE Bulletin* 36.1, pp. 75–79.

Pokress, SC and JJD Veiga (2013). "MIT App Inventor: Enabling personal mobile computing". In: *Workshop on Programming for Mobile and Touch*.

Resnick, Mitchel et al. (2009). "Scratch: programming for all". In: *Communications of the ACM* 52.11, pp. 60–67.

References III

Rivers, K and KR Koedinger (2012). "A canonicalizing model for building programming tutors". In: *Intelligent Tutoring Systems (ITS)*.

– (2013). "Automatic generation of programming feedback: A data-driven approach". In: *The First Workshop on AI-supported Education for Computer Science (AIEDCS 2013)*.

Stamper, JC et al. (2013). "Experimental evaluation of automatic hint generation for a logic tutor". In: *Artificial Intelligence in Education (AIED)* 22.1, pp. 3–17.