# Grasshopper's Event System

Defining and reacting to noteworthy features of student code

Yana Malysheva
Google, Inc.
San Francisco, CA 94105
Email: yanamal@google.com

***Abstract*—Grasshopper is an Android application which teaches users JavaScript through a series of coding puzzles. Grasshopper is able to make two types of real-time decisions based on the user's current performance: selecting an appropriate piece of feedback when the student is in the middle of solving a puzzle; and selecting the most appropriate next puzzle when the student is done with the current puzzle. For both of these decisions, Grasshopper relies on "student events": definitions of things that are noteworthy when present in student code.**

**This poster presentation illustrates the student event system, and how it is used to make these decisions, through an example of a hypothetical student going through the process of selecting and solving a single puzzle ("Dash of Random"). It also briefly demonstrates how the events are used to define the space of possible user knowledge states, and how the Dash of Random puzzle fits into that space.**

## I. Introduction

Grasshopper teaches programming through a series of coding puzzles (which we call "cards"), which the user solves by composing code in an AST-based block-like coding interface. A card normally starts off with a partial solution, and asks the user to complete or correct it to fulfill a specific requirement. Each card introduces at most one new concept, and the cards build on each other to form an understanding of the basic programming concepts.

Each time the user completes or abandons a card, the system selects and suggests the next card for the user to try. As the user is solving a card, the system selects the most appropriate hint or feedback to surface to the user.

This presentation demonstrates how these selections are made through observing a hypothetical example user who has just completed the first card (French Flag).

Table I lists some of the relevant events that are tracked and used by the system to make these decisions.

## II. Selecting a Card

After completing the very first card, our hypothetical user has been exposed to two events: `drawBox` and `newLine` (because the first card uses the functions `drawBox()` and `newLine()` to draw a flag). The system has also noted that this user completed the previous card on the first attempt, without being shown any hints. So for the next card, it will try to select a card with high complexity.

For a user with only `drawBox` and `newLine`, two cards are available: "Dash of Random" builds on `drawBox` and introduces the idea of picking random colors (`pickRandom` event); and "Short Rainbow" introduces a shortcut function, `drawBoxes()`, that can draw several boxes with one command (`drawBoxes` event).

All other cards are locked, because they introduce more than one additional event that the user has not seen yet.

Actually, "Dash of Random" would also be locked, if not for some overrides:

- It technically introduces an additional event, `callNesting`, but that event is always ignored for the purposes of card unlocking (`ignoreAsPrereq` flag)
- It also doesn't explicitly show the `pickRandom()` function in its starter code, which would normally mean that the system wouldn't use it to introduce this event; but the card creator has added a manual override that explicitly lists this card as a good card to introduce `pickRandom()`.

These two overrides mean that the Dash of Random flag is more complex than Short Rainbow: a card's complexity for a given user is a weighted sum of all the events in that card, with the weights coming from the user's familiarity with each event. In this case, Dash of Random has complexity 6.4, and Short Rainbow just 1.2.

Because our user solved the previous card very quickly, the system decides to suggest the more complex card to the user.

## III. Solving the Card

The Dash of Random card starts the user off with the code:

```
drawBox(blue)
drawBox(red)
```

The expected solution code is something like:

```
drawBox(pickRandom(color))
drawBox(pickRandom(color))
```

Each time the user runs a modified version of the code, Grasshopper tracks the subset of relevant events that are triggered by that code: relevant events are all the global events, plus events that are specific to that card. The card-specific events are usually geared toward providing hints when the solution is not quite right. For example, in Dash of Random, the event `noPickRandom` triggers when there's no `pickRandom()` call at all - in which case the user is prompted to add `pickRandom(color)` to their code.

978-1-5386-2480-7/17/$31.00 ©2017 IEEE

Table I
EXAMPLE EVENTS IN GRASSHOPPER.

| Event | Scope | Triggers when... | displayOften? | ignoreAsPrereq? |
|---|---|---|---|---|
| drawBox | global | The drawBox() function is called | | |
| pickRandom | global | The pickRandom() function is called | | |
| newLine | global | The newLine() function is called | | Yes |
| pickRandomStatement | global | The pickRandom() function is called as a stand-alone statement | Yes | |
| greebRatherThanGreen | global | The string literal "greeb" is present in the code | Yes | |
| callNesting | global | There is a nested function call like foo(bar()) | | Yes |
| noPickRandom | Dash of Random | There is no pickRandom call in the code | Yes | |
| notOneLine | Short Rainbow | There's more than one line of code | Yes | |

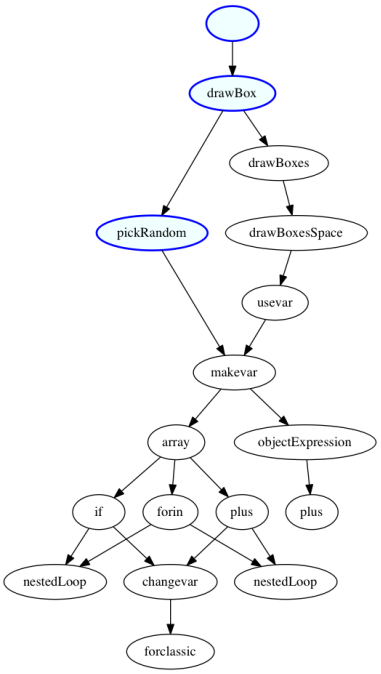


Figure 1. A graph of dependencies between events, as inferred from the set of cards in the system. The events already unlocked by our hypothetical user are highlighted in blue.

There is also a global event (`pickRandomStatement`) for when `pickRandom()` is called as a stand-alone statement, without using its return value in any way. This is almost always wrong, so the feedback for that is relevant even outside of Dash of Random. Unlike most global events, `pickRandomStatement` is flagged as `displayOften`, so the associated feedback will be shown as often as possible.

## IV. LEARNING SPACE

We can generate the space of all possible user states by reasoning recursively about which cards are available at any given state, and for each card, what new state would become available if the user completes this card.

We can summarize this giant space of possibilities, called a learning space [1], through a partially ordered set of key states, and the associated events.

Thus, even though each card is defined in a stand-alone way and does not make any assumptions about or references to other cards, we can automatically map out the flow of the curriculum that is created by the set of all cards, as in Figure 1

## V. RELATED WORK

There is a lot of prior work around using event-like metrics to evaluate student code for the purpose of providing feedback or assessing student performance: Fields, Quirke and Amely [2] developed meaningful quantitative measures of student use of programming concepts across several projects in a constructionist camp environment. Ichinco and Kelleher [3] have advocated for a system where mentors or experts author rules that check for specific common issues in student code and trigger associated feedback. Dr. Scratch [4] and Dr. Pencil Code are web tools that analyze a student's Scratch or Pencil Code program and give them direct feedback in several Computational Thinking-related categories. Pencil Code Gym [5] is a set of programming tutorials which, when appropriate, provide feedback in the context of the current goal or topic.

Separately, there are also some very interesting machine learning approaches to the problem of analyzing and reacting to student code, such as clustering partial solutions into meaningful states and then analyzing how students progress through them [6]; and predicting an optimal "next state" for a student to move toward, given their current state, by inferring problem solving policies from data on how other students have solved the problem [7].

## REFERENCES

[1] J.-C. Falmagne and J.-P. Doignon, *Learning spaces: Interdisciplinary applied mathematics*. Springer Science & Business Media, 2010.
[2] D. A. Fields, L. C. Quirke, and J. Amely, "Measuring learning in an open-ended, constructionist-based progamming camp: Developing a set of quantitative measures from qualitative analysis," in *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*, Oct 2015, pp. 15–17.
[3] M. Ichinco and C. Kelleher, "Online community members as mentors for novice programmers position statement," in *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*, Oct 2015, pp. 105–107.
[4] J. Moreno-León and G. Robles, "Dr. scratch: A web tool to automatically evaluate scratch projects," in *Proceedings of the Workshop in Primary and Secondary Computing Education*, ser. WiPSCE '15. New York, NY, USA: ACM, 2015, pp. 132–133. [Online]. Available: http://doi.acm.org/10.1145/2818314.2818338
[5] "Pencil code gym," 2014. [Online]. Available: https://gym.pencilcode.net/
[6] C. Piech, M. Sahami, D. Koller, S. Cooper, and P. Blikstein, "Modeling how students learn to program," in *Proceedings of the 43rd ACM technical symposium on Computer Science Education*. ACM, 2012, pp. 153–160.
[7] C. Piech, M. Sahami, J. Huang, and L. Guibas, "Autonomously generating hints by inferring problem solving policies," in *Proceedings of the Second (2015) ACM Conference on Learning@ Scale*. ACM, 2015, pp. 195–204.