Table of Contents

λ - An Autograder for Snap*!*

This is the repository for my Master's Thesis, formatted as a GitBook.

You can view the online version (WIP).

Title	**λ - An Autograder for Snap*!***
Author	Michael Ball
Advisor	Dr. Dan Garcia
Degree	Masters of Science, Computer Science
Emphasis	Computer Science Education
Date	May 2016

Abstract

While visual programming languages are hardly a new development, recent pushes for increased equity and access to computer science have led to a renewed interest and use of visual programming languages. Autograders are a critical component of both in-person and online computer science courses. However, until this point, there haven't been any autograders available for these environments. This is a particular shortcoming as introductory courses have scaled to larger numbers of students and online environments. While there are challenges to using autograders, we believe that the instant feedback capabilities, as well as potential time savings for course staff will help us teach a greater number of students.

Over the past year, we have built an autograder, named λ (Lambda) for Snap*!*, a visual programming language inspired by MIT's Scratch. The primary motivation for developing the autograder was to run a series of Massive Open Online Courses (MOOCs) on edx.org throughout the 2015-2016. However, we also would like to use the autograder to better support in-person computer science courses. In Spring 2016, the Snap*!* autograder was used as a part of UC Berkeley's CS10, *The Beauty and Joy of Computing*, a "CS0" non-majors course.

This report describes λ which consists of a "backend" Ruby on Rails webserver that allows us to use the autograder in a classroom setting, through a protocol called LTI. The backend web application contains at database of questions and test files, while the Snap*!* interface contains new features as well as view to present the results of the autograder. Our initial results show the autograder successfully being used in CS10 where the autograder was used to supplement oral lab checkoffs, and on edx.org where the autograder was the primary method for students to receive credit for code.

Acknowledgements

TODO:

- Dan
- Snap!
- BJC
- Lauren
- CS10

List of Figures

Introduction

In recent years, the field computer science (CS), and the broader technology industry, have undergone significant changes, though not simply technical ones. Around the same that same time from that enrollments started booming [Citation `'cra-blog'` not found], the National Science Foundation (NSF) posted the original solicitation for *Broadening Participation in Computing* [1]. As a result of the efforts to improve diversity in CS, curricula such as *Exploring Computer Science* [2] (ECS), and *AP Computer Science Principles* [3]. (AP-CSP) have emerged. A central component of these new curricula includes the ability to use visual programming languages (VPLs) to teach CS. As great as these tools can be to teach programming, the often lack infrastructure found in more traditional environments (such as Python and Java). This lack on infrastructure can make courses more difficult to teach and scale [Citation `'REF TODO'` not found].

A critical example of missing infrastructure is the capability to automatically evaluate code, and send the results to another process. Automatic evaluation, and distributing the results are central components in an *autograder*. Without an autograder, it can be difficult (or simply costly) teach large online courses. During the 2015-2016 school year, we taught *The Beauty and Joy of Computing* [4] (BJC) a series of four *massive open online courses* (MOOCs) on edX [Citation `'edx'` not found]. BJC is an AP CS Principles course that uses Snap*!* as its primary language. Without an autograder, we didn't think that we could fairly give credit to students online. In addition, BJC is offered at UC Berkeley as CS10 would be able to make use of an autograder for Snap*!*.

We developed λ as a system for autograding Snap*!* [5]. λ is composed of two main parts, the Ruby on Rails application server, and a Javascript application that augments Snap*!* with testing, analysis and logging capabilities. While the grading interface was developed for edX, we report on its use in the classroom and some recommendations for future development. We also hope that λ will be useful outside CS10, through the implementation of the Learning Tools Interoperability (LTI) protocol which should allow our system to be compatible with most Learning Management Systems. We've built additional features which should allow flexibility to integrate the autograder into a variety of classroom environments. These features are described in the chapter on implementation.

Snap*!*

Snap*!* is inspired by MIT's Scratch [Citation `'scratch'` not found], but adapted for university-level courses by including features such as first-class lists and functions. Snap*!* is implanted as a web application in Javascript.

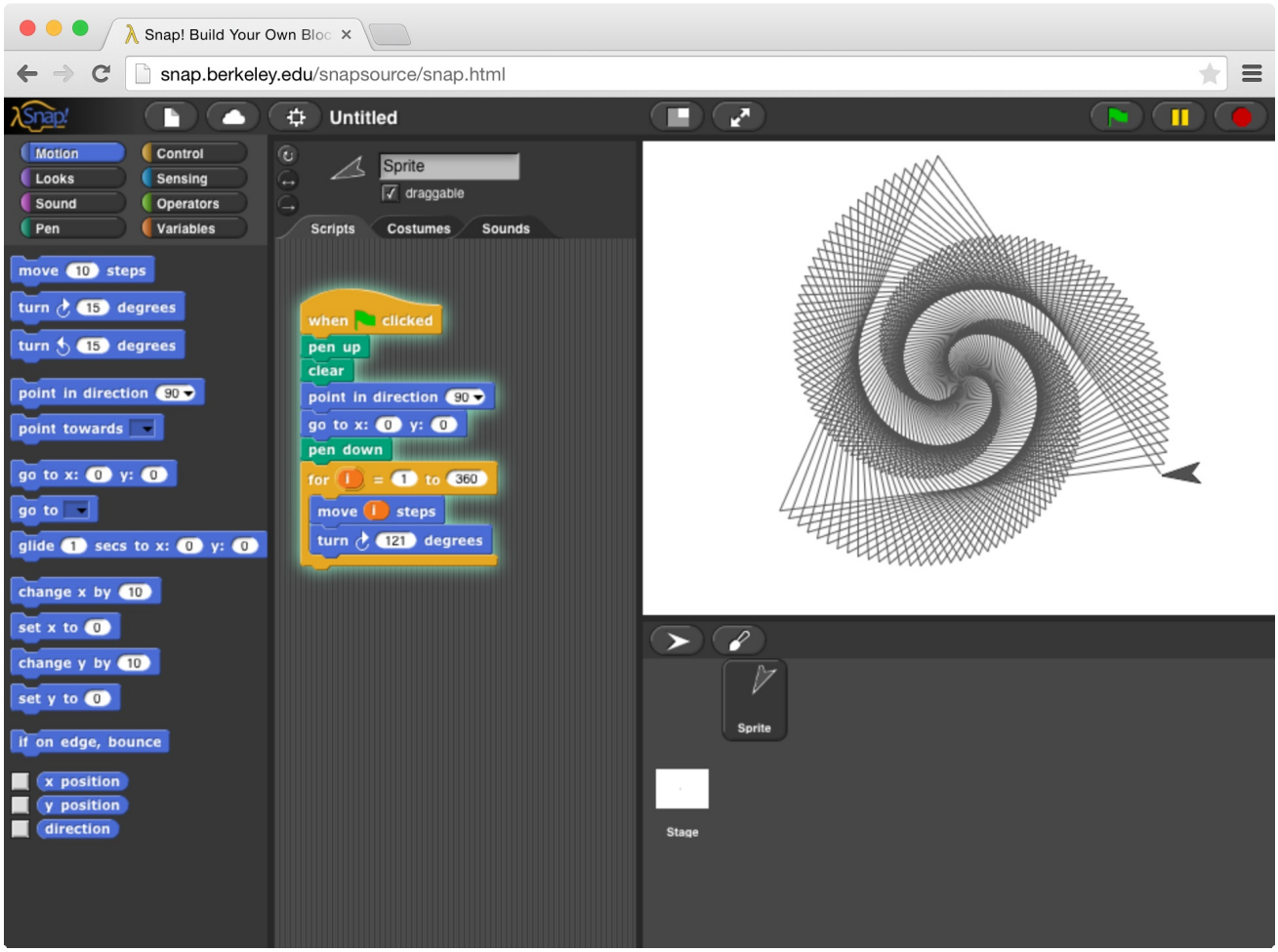

Figure 1: *An example Snap! program.*

The Beauty and Joy of Computing

The Beauty and Joy of Computing is a introduction to computing designed to broaden participation among underrepresented groups. The primary language used is Snap*!*, and many of the exercises in the course have visual outputs. The course functions and abstractions, covers recursion, higher order functions (lambdas) and many other topics. Many of the examples and exercises in BJC have multiple paths to implementation, which can be a challenge for autograders to correctly handle.

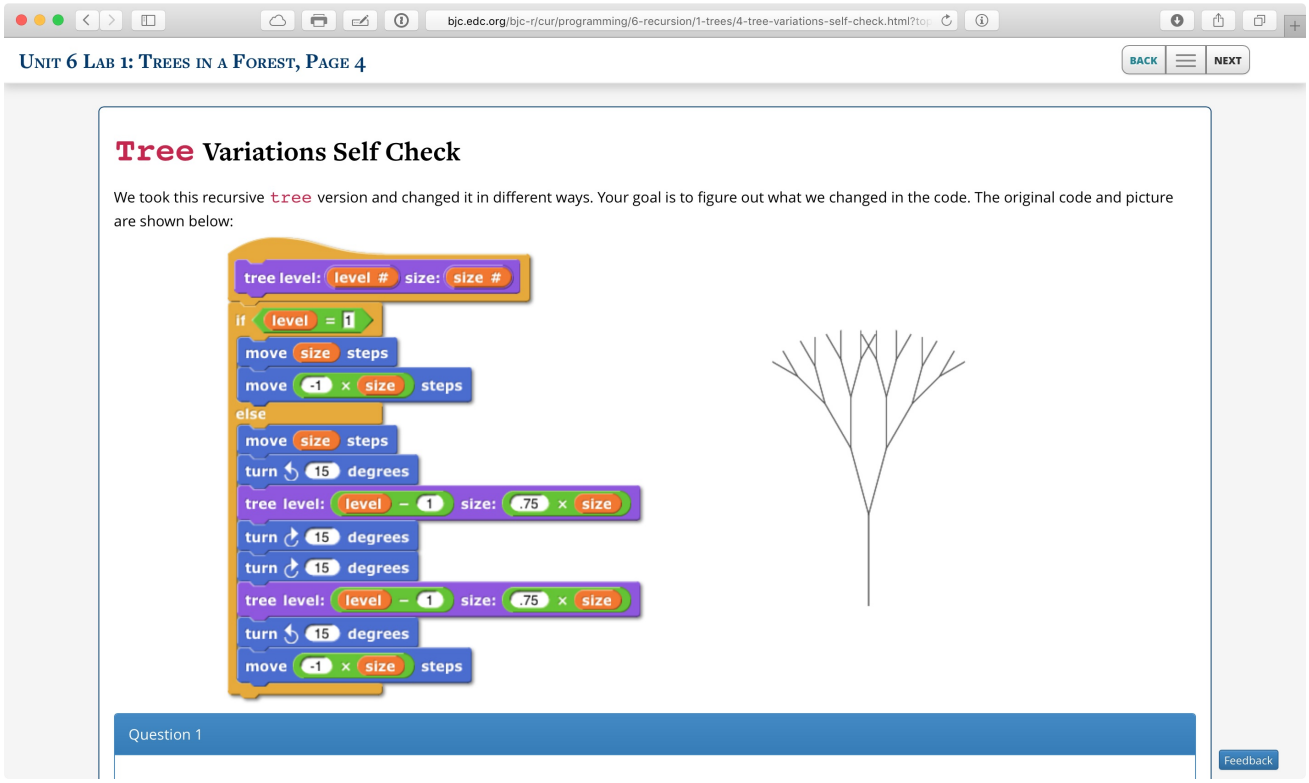

Figure 2: *A typical example of BJC curriculum which includes graphical output.*

CS10

CS10 [6] is UC Berkeley's offering of BJC. Currently, CS10 is offered every semester to around 200-300 students. Like many other introductory courses, CS10 relies heavily on laboratory sections as the primary method for students to learn to program. (However, programming is of course covered in lecture and discussion or 'recitation' sections.) λ will be used in CS10 during lab sections to give students better feedback as they're working as well as to give credit for the assigned lab work. This year, we were able to trial the system for three of the 14 labs that students completed in Snap*!*.

Related Work

In building and designing an autograder, there is surprisingly little work published about autograders, and even less so about those for visual programming languages. While there are many examples of autograders for languages like Python and Java, there are only two that we are aware of for blocks-based languages, which are both very recent developments.

Code.org CodeStudio

Code.org's CodeStudio [Citation `'code-studio'` not found] is an online interactive environment based around the open source Blockly environment. [7] CodeStudio includes custom feedback, but the vast majority of the problems are in a highly restrictive space where students are given only a limited set of blocks. While CodeStudio is open source [Citation `'code-studio-github'` not found], the autograder and feedback methodologies are not documented.

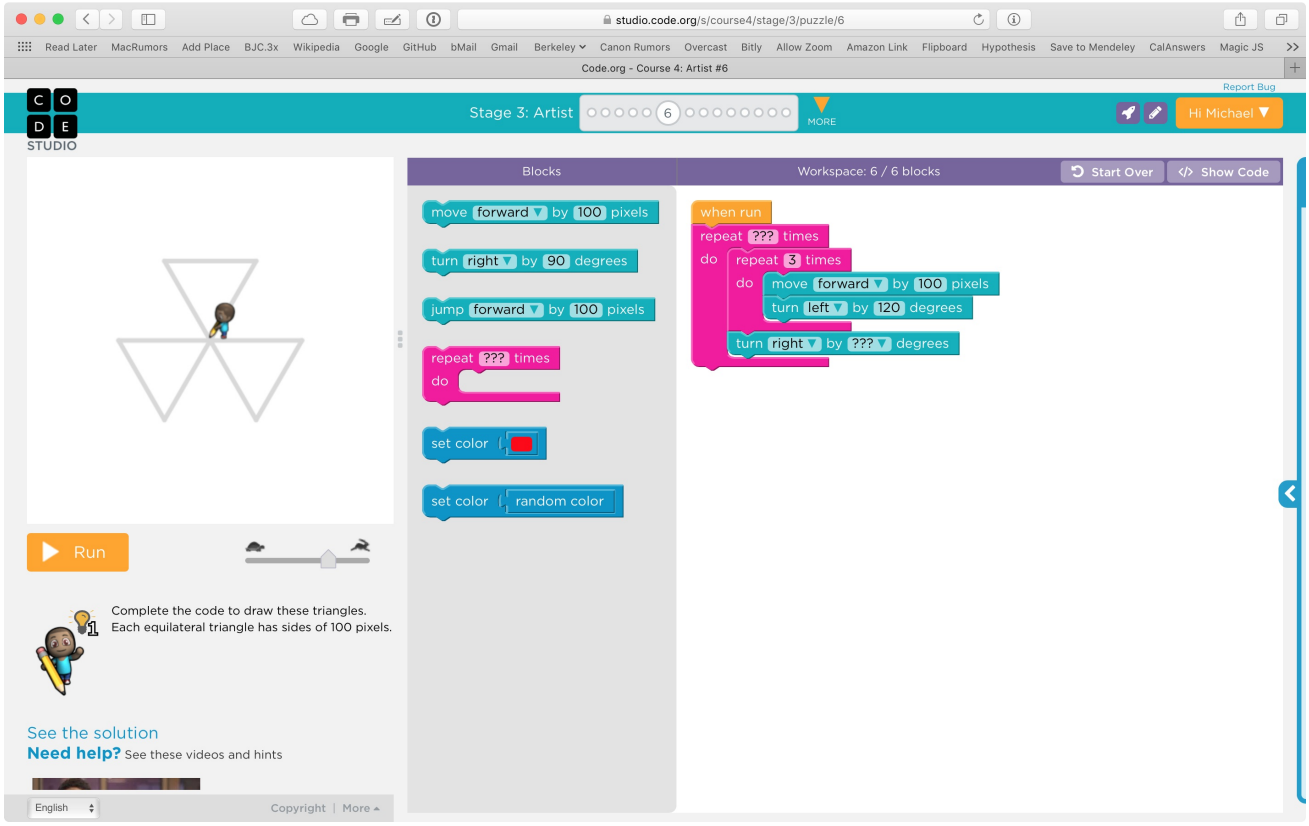

Figure 3: *A typical example of a CodeStudio problem that gives students only a few blocks to work with, and has a fairly constrained solution space.*

ITCH: Individual Testing of Computer Homework for Scratch Assignments

ITCH is an autograder for Scratch recently presented at SIGCSE 2016 [Citation `'itch'` not found]. However, ITCH's method of autograding is through an instructor-initiated script which should be run after all assignment submissions are collected. This is a fairly common scenario for typical code autograders, but not a model we would use because our goal is instant feedback. ITCH takes a similar method to the Snap*!* autograder by 'spoofing' user inputs when requested, but it shares many of the same limitations, like a lack of feedback for graphical output.

Design Principles

The design of the autograder is influenced by a number of design principles.

Knowledge Integration

Learner Centered Design

Constructionism

Implementation

The λ web application is built using the Ruby on Rails framework [Citation `'ror'` not found], and makes use of many common web technologies. The Snap*!* interface is implemented purely in JavaScript.

The Need for a Web App

The JavaScript which powers the autograding works entirely client-side, meaning as long as you have the test files, there's no need for an internet connection. This path was chosen for two reasons:

- Snap*!* is client-side, and evaluating Snap*!* projects on a server would require a significant amount of work.
- edX provides a custom problem type called "JSInput" [Citation `'jsinput-docs'` not found] which gave us a clear path for integration with edX.
- Developing an autograder and a web application required more resources than were available.

While the entirely client-side path was a good decision, we ran into a number of issues by relying on `JSInput` and trying to keep all features client-side.

`JSInput`

edX's `JSInput` problem type provides a JavaScript API for sending scores to the edX platform. It allows us to build in a custom version of Snap*!* alongside the rest of the content in edX.

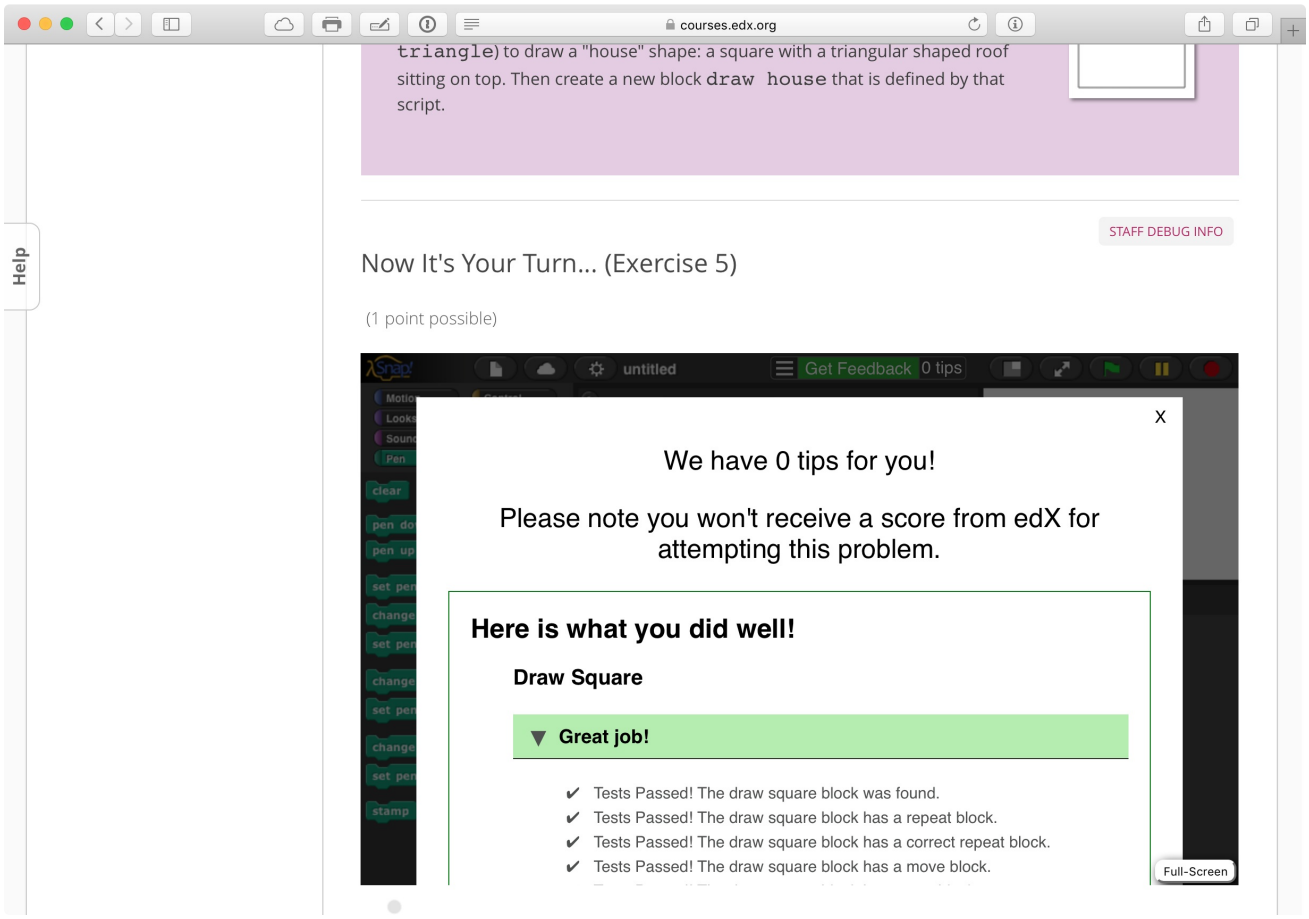

Figure 4: *Snap! can be embedded in edX through JSInput.*

However, we encountered several problems while developing the `JSInput` based integration:

- Developing code was very slow! Changes to code required manually uploading a new file to edX, which is a multi-step process. The libraries used for JSInput swallowed native JavaScript errors, making debugging nearly impossible.
- The edX interface has it's own mechanism for a "Check" button and showing feedback. Communicating the detailed output from the autograder didn't work very well, and we ended up developing many work arounds to get a seamless UI.
- There's no room for storing or retrieving user metadata. We rely on features that allow students to recall previous submissions. Through `JSInput` the only option for these features were to use the browser's `LocalStorage` API. This API has limits, like a max of 5MB of storage, that caused problems for some students.
- Furthermore, without a dedicated database, every single test file written had lots of hard-coded metadata that was repetitive and prone to error.

- While edX provides user logs for the entire course, we found dealing with these logs to be needlessly complex. They are slow to get, and the autograder results are difficult to separate from the rest of the course data. As such, we haven't analyzed the edX data to the extent we'd like to. A simpler logging system described below has been immensely helpful for our analysis.

The one benefit of these problems was that it forced the development of the autograder into two components: A JS interface to edX, and a "dumb" client-side component that sits on top of Snap*!*. This distinction was helpful when adapting the autograder to work with the new web application.

perhaps most importantly: the grading system could only work with edX. CS10 uses Canvas [Citation `'canvas'` not found] as it's LMS, and many high schools use different systems. The need to build a custom solution for every platform would be prohibitively expensive. → LTI

Basic Architecture

- Rails
- Postgres
- Snap*!*

LTI & User Accounts

- LTI
- Oauth
- Snap*!*
- User mapping

Security Concerns

...

```
* edX JS Input
* Moving to LTI
* Dedicated Database
    * Simpler JS Test
```

Trail Setup

Results

Conclusion

Coming Soon.

References

1.	Cuny, Janice and NSF, nsf09534 Broadening Participation in Computing (BPC)	NSF - National Science Foundation, 2009. https://www.nsf.gov/publications/pub_summ.jsp?WT.z_pims_id=13510&ods_key=nsf09534
2.	Exploring Computer Science, http://www.exploringcs.org/
3.	Board, The College and Diez, Lien, AP Computer Science Principles - A New AP Course - Advances in AP - The College Board	Advances in AP, https://advancesinap.collegeboard.org/stem/computer-science-principles
4.	Harvey, Brian and Garcia, Daniel and Development, Corporation Educational, BJC - Beauty and Joy of Computing, http://bjc.berkeley.edu/
5.	Harvey, Brian and Mönig, Jens, Snap! (Build Your Own Blocks) 4.0, 2016. http://snap.berkeley.edu/
6.	{UC Berkeley}, {CS10	Spring 2016}, http://cs10.org/sp16/
7.	Fraser, Niel, {Blockly	Google Developers}, https://developers.google.com/blockly/