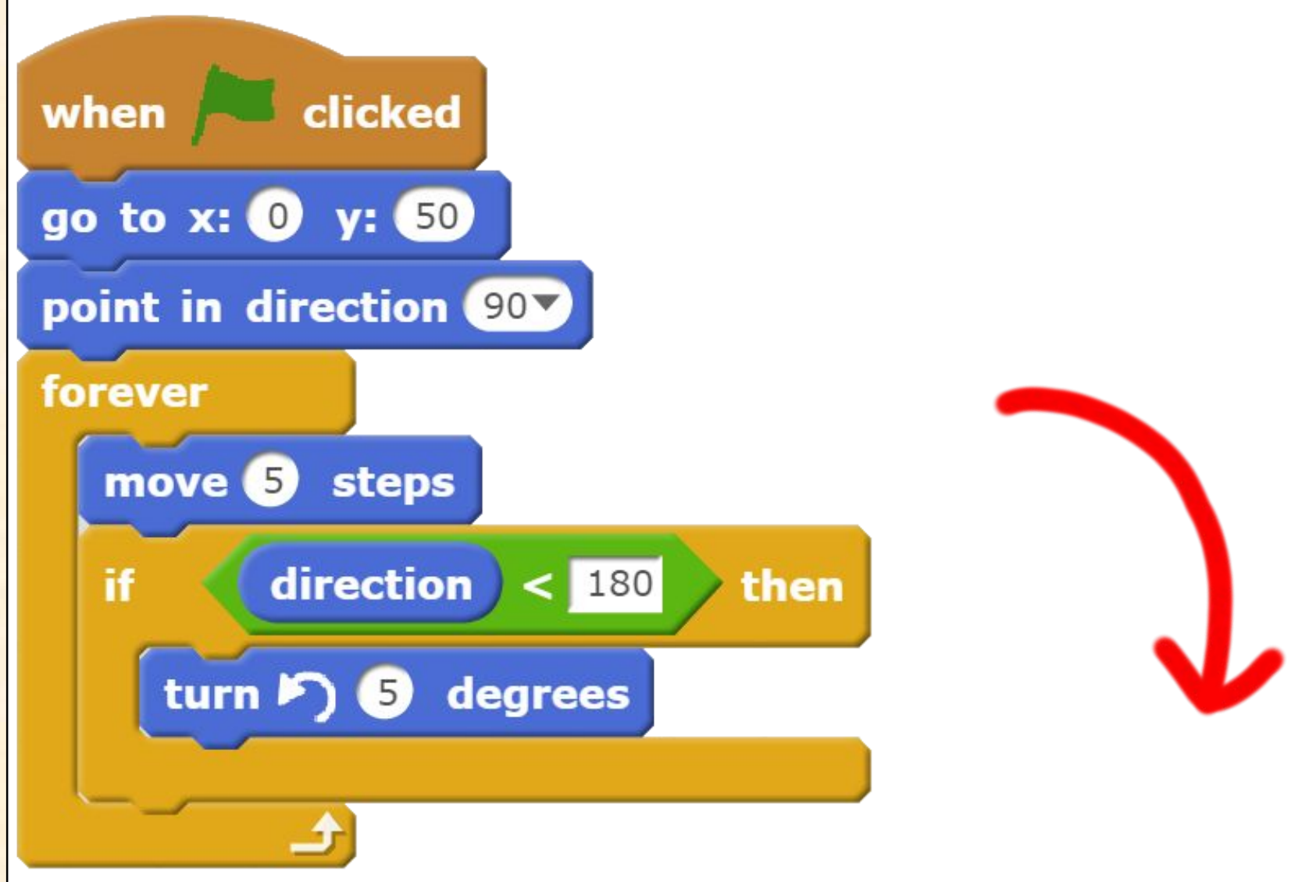
ScratchFoot: Transitioning New Programmers from Blocks to Text

Victor Norman, Jordan Doorlag

Calvin College 1876

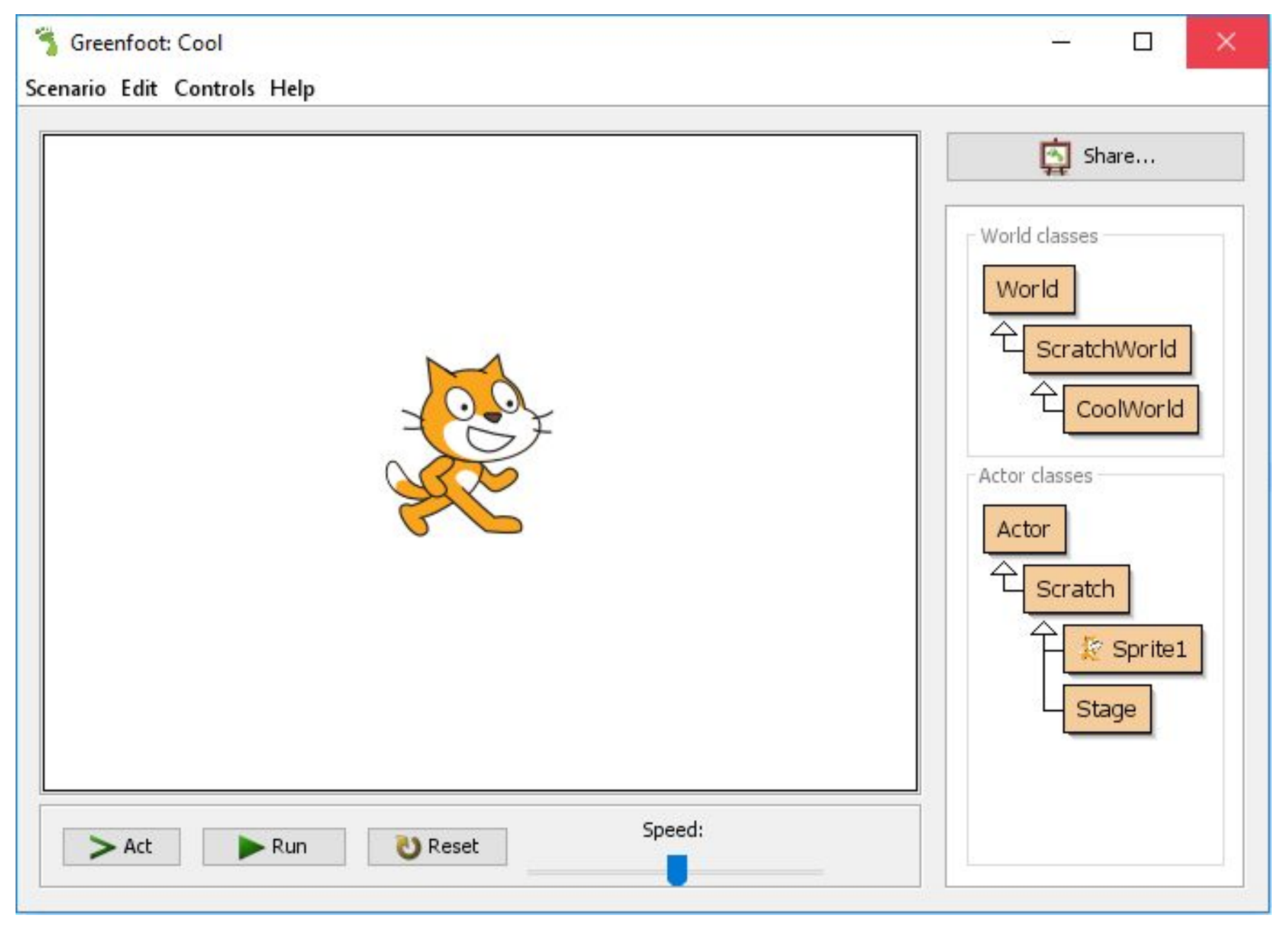
Abstract

When a child is first learning to code, she will often use a highly-controlled, drag-and-drop style environment such as Scratch. These environments are great because students learn the fundamentals of programming without having to deal with many of the syntactic complexities that come with a text-based language. However, students have difficulty transitioning from these block-based environments to the “real world” of text-based languages. The goal of this project is to ease this transition with a program that can convert Scratch blocks into Greenfoot Java code that closely resembles the Scratch blocks. The program consists of two parts: an API for Greenfoot which maps each Scratch block to a line of Java code, and a separate program to convert a Scratch project into Greenfoot code, images, and sounds.



```
public void whenFlagClickedCb0(Sequence s)
{
    goTo(0, 50);
    pointInDirection(90);
    while (true)        // forever loop
    {
        move(5);
        if (getDirection() < 180)
        {
            turnLeftDegrees(5);
        }
        yield(s);   // allow other sequences to run
    }
}
```

Scratch in Greenfoot

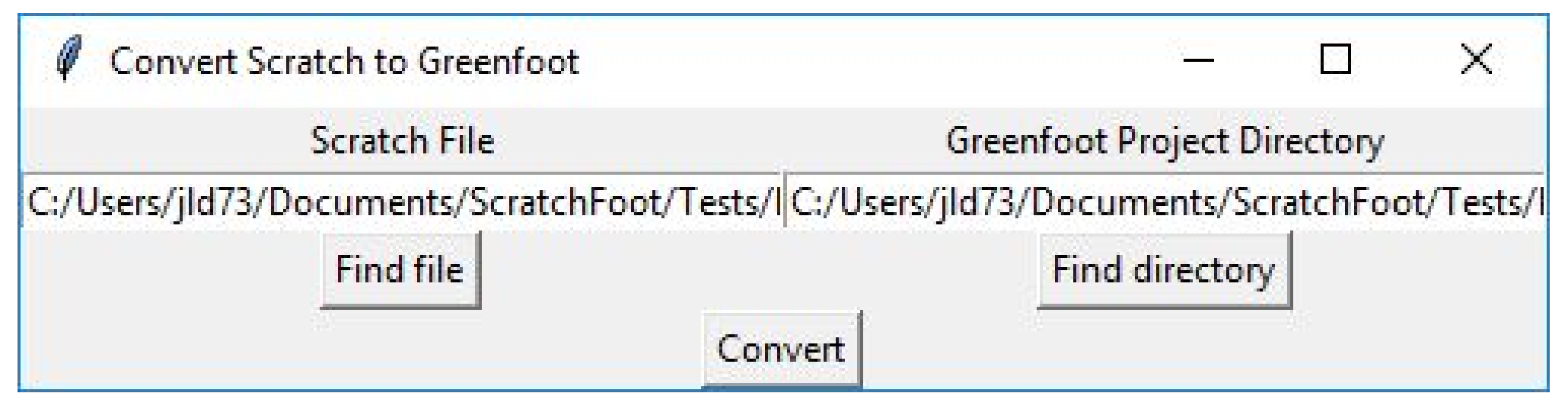


The Greenfoot UI, with the ScratchFoot class hierarchy displayed on the right

The ScratchFoot API consists of two Java classes, Scratch.java and ScratchWorld.java. These classes serve as superclasses for all objects in the scene, providing a collection of methods which implement the functionality of Scratch blocks in the Greenfoot/Java environment. An important goal was to keep the API simple so that students can easily use the API and can recognize their Scratch code in its Greenfoot/Java form. However, implementing the API came with some difficulty, as Scratch uses untyped variables and has multiple scripts which execute (seemingly) in parallel, while Greenfoot/Java has strict types and sequential execution. Thus, keeping the interface simple occasionally meant implementing the API using complex data structures and algorithms.
Additional challenges included reimplementing in Java some Scratch blocks that rely on core features of Flash. We have, however, managed to get a simulacrum of every block working, except for a few that are especially complicated and rarely used.

Conversion

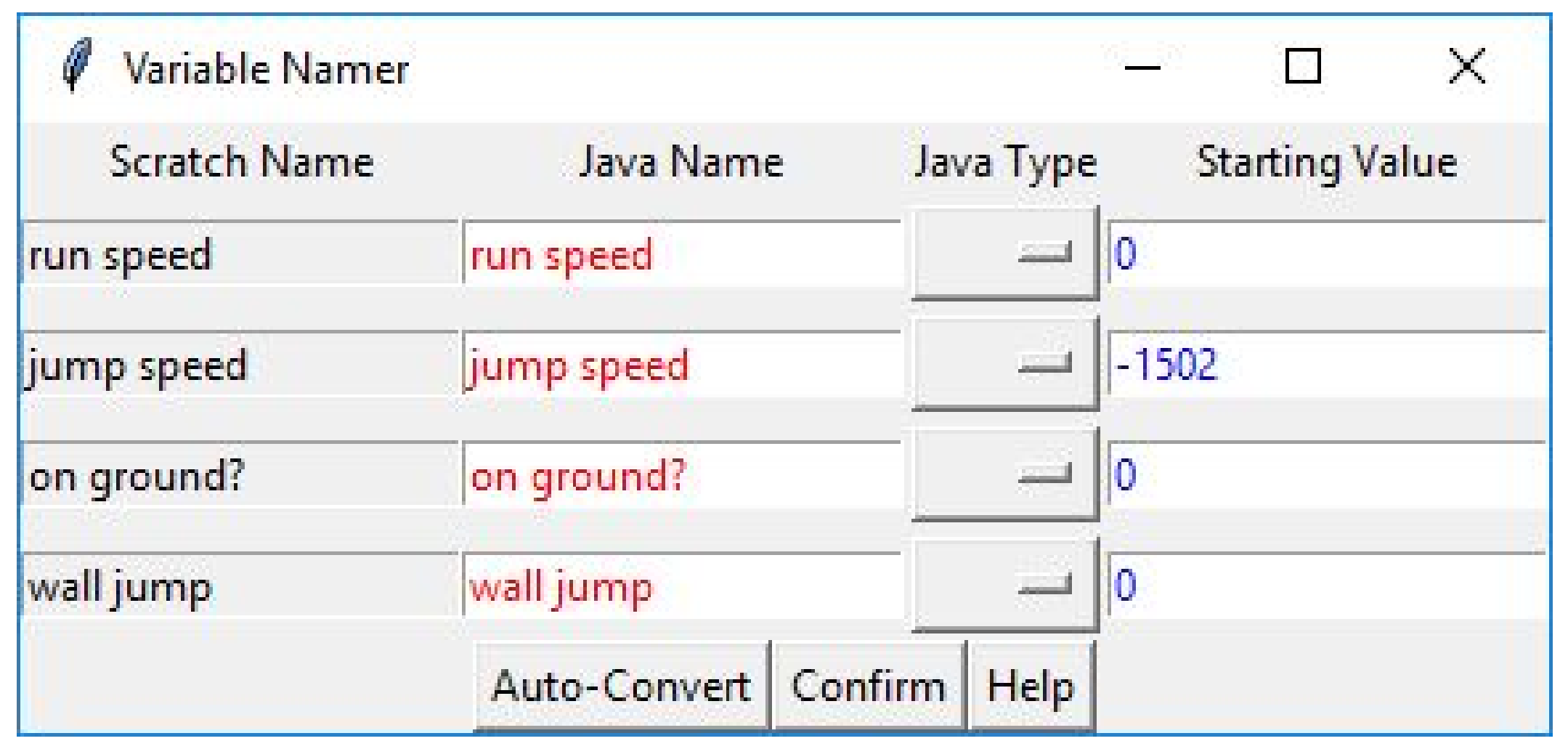
The second part of this project is the Scratch to Greenfoot converter. This is a python script with a simple GUI that parses a downloaded Scratch file, converting the data found in the JSON file into Java classes.



Convert Scratch to Greenfoot

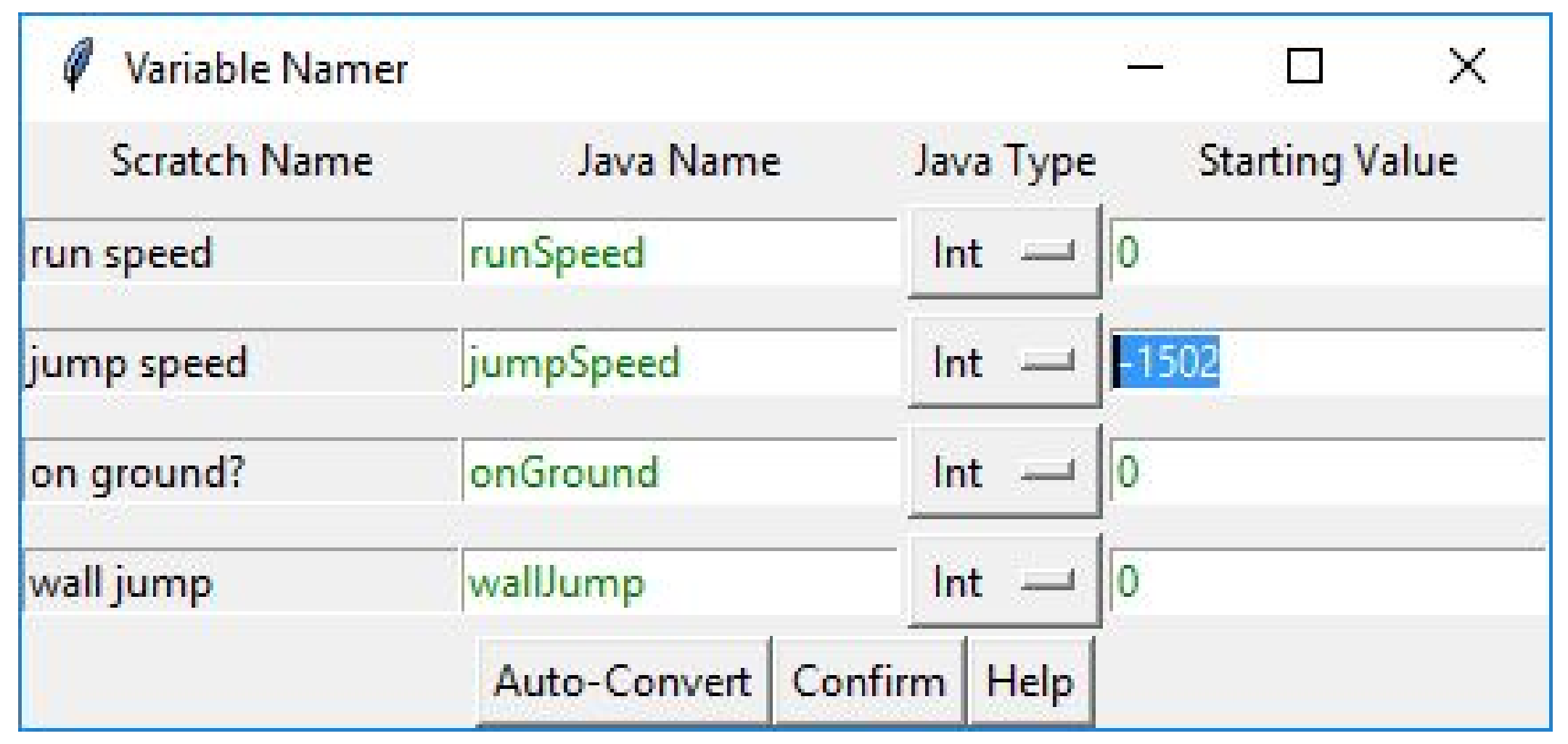
Scratch File	Greenfoot Project Directory
C:/Users/jld73/Documents/ScratchFoot/Tests/I	C:/Users/jld73/Documents/ScratchFoot/Tests/I

The conversion is complicated because Scratch allows any character or symbol to be used in variable and object names, whereas Java’s rules are much more strict. To solve this, we developed a GUI to allow the user to rename and choose the type of every variable in the project. To improve usability, especially among those who aren’t familiar with Java, a feature was added to automatically convert the names to Java identifiers and offer a suggestion for the type.



Variable Namer

Scratch Name	Java Name	Java Type	Starting Value
run speed	run speed		0
jump speed	jump speed		-1502
on ground?	on ground?		0
wall jump	wall jump		0



Variable Namer

Scratch Name	Java Name	Java Type	Starting Value
run speed	runSpeed	Int	0
jump speed	jumpSpeed	Int	-1502
on ground?	onGround	Int	0
wall jump	wallJump	Int	0

Ongoing Work

We are currently rewriting the conversion program. The initial implementation used a recursive descent parser, a relatively simple process that looks at each block in the Scratch scripts, generating code as it recognizes each statement. The rewrite still uses a recursive descent parser, but generates an Abstract Syntax Tree instead. The AST is a data structure that represents everything in the program. The conversion program analyzes the AST as necessary to make more accurate predictions, such as determining the type of a variable based on which blocks use it. This makes the generation process much cleaner, and separates code generation from parsing. It also allows future modifications to generate code for other languages (Python with Tkinter) or programming environments.

The Future

There are several things that we would like to do in the future.

- Live testing with Middle to High school students.
- Optimize the API.
- Improve the GUI to provide more helpful feedback.
- Develop a curriculum using ScratchFoot to transition kids away from the ‘forever loop’ mentality of Scratch.