Musings on Teaching Basic Programming Concepts

Warren Toomey

This is a short look at some of the drawbacks of teaching Java as a 1st programing language, some alternatives, and the obstacles in the way of teaching basic programming skills.

Questions and short discussions along the way are most welcome.

Why Do We Teach Java?

- Reasonably abstract (i.e. better than C, assembly language)
- Good collection of control structures: if, for, while, methods etc.
- Good data structures: objects, arrays, object references
- Available on nearly all common platforms
- A popular language, reasonably standard, not too proprietary

What's Bad About Java as a 1st Programming Language?

- So much structure to learn:

```
public class AccountList
public static void main(String [] args)
Scanner scan = new Scanner(System.in);
```

- Only basic interactivity:

```
scan.nextInt();
System.out.println();
```

- Objects and types get in the road of doing things
- Graphics are not easy to do
- Edit, compile, run cycle: results are not seen immediately by the student

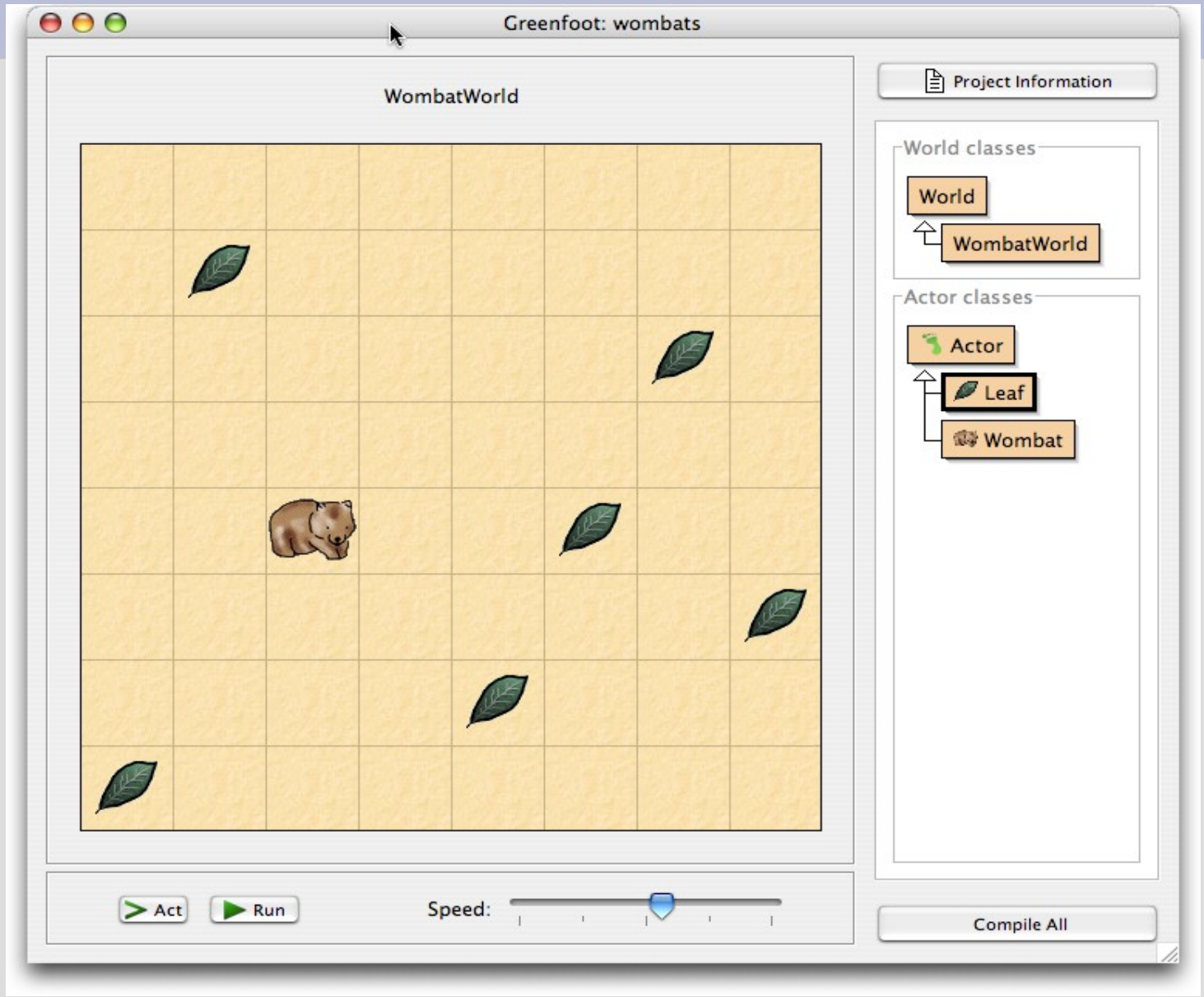
Can This Be Overcome?

- Can we hide the language baggage, while providing good control and data structures?
- Can we provide more interactivity, so as to engage the students and make the learning experience more enjoyable?
- Should we start with a simpler language before progressing to Java or an equivalent?
- Let’s look at some other languages/tools

Greenfoot

- Embeds the Java language in a graphical environment
- Instances of actor classes can be dropped into the world
- Each instance can be controlled manually, or the world “run” so that all actors show their behaviour
- There is access to the Java code for each actor class: to view, to modify, to create new actor classes
- *www.greenfoot.org*

Greenfoot


Greenfoot: wombats
WombatWorld
Project Information
World classes
World
WombatWorld
Actor classes
Actor
Leaf
Wombat
Act
Run
Speed:
Compile All

Greenfoot

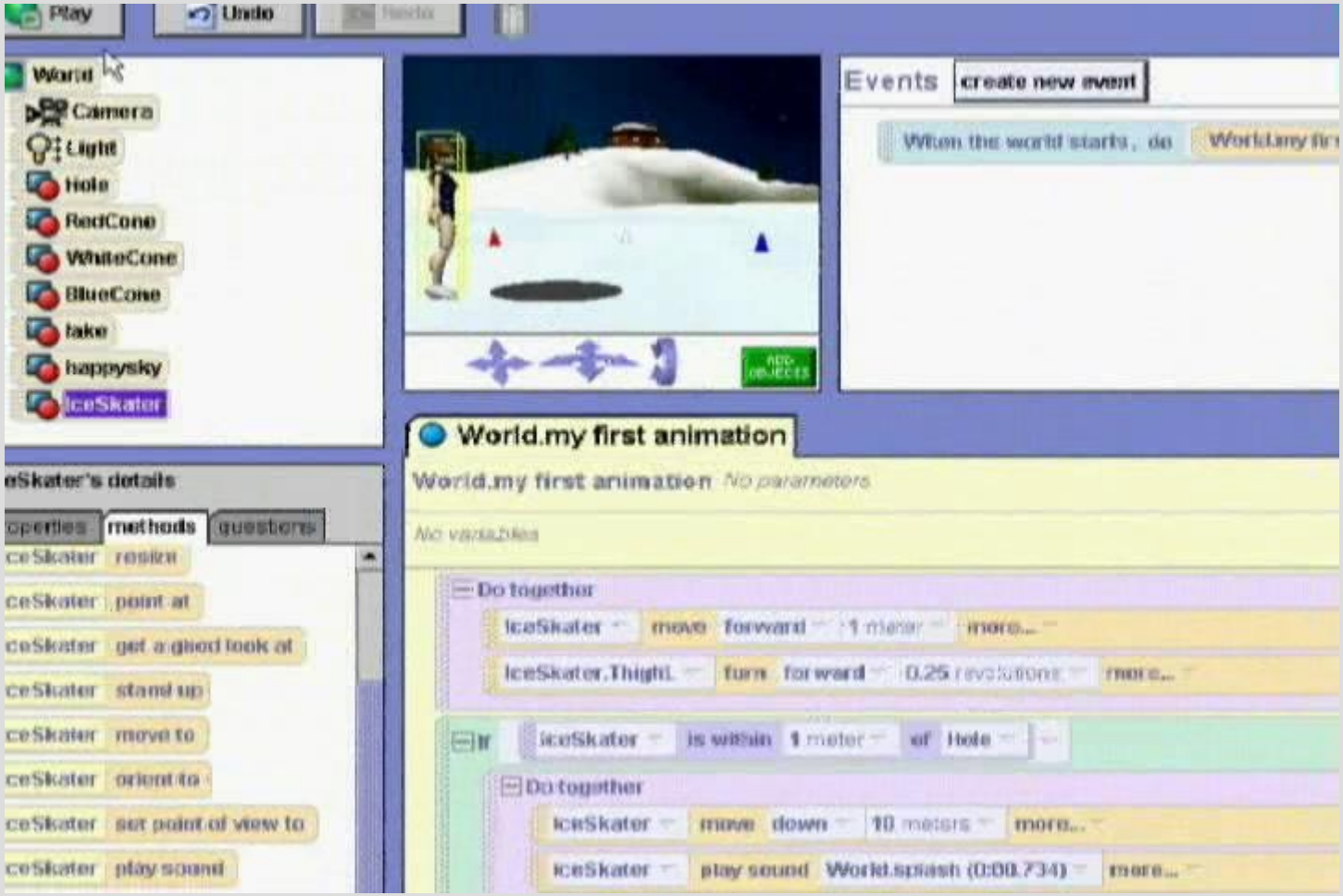

inherited from Object
inherited from ObjectTransporter
inherited from Actor
void act()
boolean canMove()
void eatLeaf()
boolean foundLeaf()
int getLeavesEaten()
void move()
void setDirection(int)
void turnLeft()
Inspect
Remove

Alice

- Like Greenfoot, Alice embeds a Java-like language in a graphical environment
- Instances of actor classes can be dropped into the world
- Each instance can be controlled manually, or the world “run” so that all actors show their behaviour
- Alice, however, seems to hide the underlying language behind a “graphical” language
- *www.alice.org*

Alice Video

- Start from position 1:52



Scratch

- Like the previous two, Scratch provides a stage where actors can perform
- This time, the language is represented by graphical elements
- There are no data structures apart from simple variables
- However, there is a large amount of multimedia support
- *scratch.mit.edu*

Scratch Video

- Start from position 1:16

My Comments on Greenfoot, Alice and Scratch

- Greenfoot: still the Java language baggage, and the framerate is slow
- Alice: seems to be somewhat maths oriented: (x,y) positions, movement etc.
- Scratch: no textual language, students will have to transition to a textual language
- Also no decent data structures, but at least the graphics & interactivity are good

Game Maker

- In 072, I taught an “Intro to Game Logic” course using Game Maker (GM)
- GM provides objects, instances, and an event model
- Programming is done either by drag 'n drop, or using a script-like textual language
- Variables are typeless, like Perl: they hold ints, floats, strings, booleans, instance-ids
- Global vars, instance vars, local vars
- 1D arrays exist

Game Maker Control Structures

- Very C and Java like.
- Expressions, assignment, IF, FOR, WHILE
- Do .. Until
- Repeat(): repeat for a given # of times
- With(): apply the following code to a specific instance, or to all instances of an object
- User-defined functions with arguments, local variables and a return value if required
- GM checks # arguments at run-time

Some Game Maker Examples

- Assignment 2: Frogger
  - In both drag 'n drop and textual form
  - The latter using student-written functions
- Some examples involving loops and the with() construct to manipulate instances
- An example of a 3D game
  - Just to show you the power of the system

Strengths of Game Maker

- Nice to lose the language baggage
- Typeless variables: easier to start with
- Students can start with drag 'n drop, and then transition to a textual language
- The overall syntax, and the control structures, are very close to C and Java
  - makes a transition to a real language easier
- Students enjoyed making games, it helped to engage them
- The built-in functions are very powerful, and make game programming easier

Weaknesses of Game Maker

- Only available on Windows (boo, hiss!)
- Some students found the transition from drag 'n drop to textual language difficult
- You cannot see “all” the code at once: it is spread around many objects, events, scripts
- The debugger has no breakpoints, no single-step
- Most difficult concepts for the students:
  - Loops
  - Functions: how to use them and when, how they work, use of local variables

What I Would Like to See

- A 26-week 1st programming subject
- Students would do 6 weeks with a scripting language like Game Maker
  - easier learning curve than Java, but still gives basic data and control structures
- Then transition to Java for more advanced stuff: recursion, inheritance, data structures etc.
- Introduce a Java-based 3D game engine mid-way through to keep their engagement levels up

Discussion Time