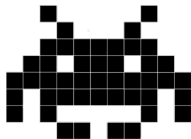


# A Teaching System To Learn Programming: The Programmer's Learning Machine

Martin Quinson, Gérald Oster  
Lorraine University, France

July 8., 2015  
itiCSE 2015, Vilnius



# The Programmer's Learning Machine

- ▶ Interactive Exerciser to Learn Programming
- ▶ Loosely tutored practical sessions at University, College

## Short Feedback Loop

The image displays three sequential screenshots of the 'Flower Pot' programming environment, illustrating a short feedback loop:

- Left Screenshot:** Shows the 'Mission' tab with 'Mission text' (a red box) and 'World view' (a red box). The 'entity' (a red box) is visible in the world view. The 'Interactive controls' (a red box) are at the bottom.
- Middle Screenshot:** Shows the 'Code' tab with the 'Code area' (a red box) containing the initial code. The 'World view' (a red box) shows a grid of colored squares.
- Right Screenshot:** Shows the 'Code' tab with the 'Code area' (a red box) containing the final code. The 'World view' (a red box) shows the completed grid. The 'console logs' (a red box) are visible at the bottom.

# The Programmer's Learning Machine

- ▶ Interactive Exerciser to Learn Programming
- ▶ Loosely tutored practical sessions at University, College

The screenshot displays the Programmer's Learning Machine interface. At the top, there is a menu bar with 'File', 'Session', 'Language', and 'Help'. Below the menu bar are several control buttons: 'Run', 'Step', 'Stop', 'Reset', 'Demo', 'Call for Help', and 'Switch exercise'. The main window is titled 'Mission: FlowerPot'. On the left side, there is a text area containing the mission description for 'Flower Pot'. The text explains that the user's goal is to reproduce a drawing of a pot and its contents, and provides instructions on how to write a `growFlowers()` method. A list of colors is provided at the bottom of the text area. On the right side, there is a 'World' view showing a grid with a green area and a small green plant. A red box labeled 'entity' points to the plant. Below the grid, there are buttons for 'forward', 'left', 'mark', and 'backward', with a red box labeled 'Interactive controls' pointing to them. The text 'World view' is also present in a red box. The text 'Mission text' is also present in a red box.

File Session Language Help

Run Step Stop Reset Demo Call for Help Switch exercise

Mission: FlowerPot

**Flower Pot**

Your buggle decided to flower a bit the pot it lives in. You have to help it reproducing the drawing of its dreams (check the "Objective" tab to see it).

For that, you have to write a `growFlowers()` method that does not take any parameter and does not return any result. As the pattern is a bit complex, you should try to decompose this drawing in sub-steps so that your code remains short and easy to read. A method to draw one flower sounds like a good enough.

For your information, this drawing uses five colors that are listed below.

- Color.RED
- Color.PINK
- Color.CYAN
- Color.ORANGE
- Color.GREEN
- Color.YELLOW

World Objective

entity

World view

forward

left mark backward

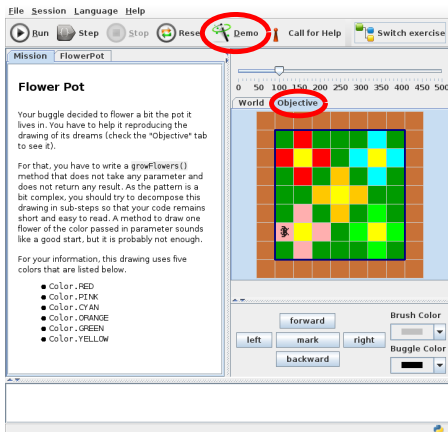
Interactive controls

Mission text

Read the mission. Inspect the initial state

# The Programmer's Learning Machine

- ▶ Interactive Exerciser to Learn Programming
- ▶ Loosely tutored practical sessions at University, College



Compare to the objective state. Check the demo?

# The Programmer's Learning Machine

- ▶ Interactive Exerciser to Learn Programming
- ▶ Loosely tutored practical sessions at University, College

The screenshot displays the Programmer's Learning Machine interface. At the top, there is a menu bar with 'File', 'Session', 'Language', and 'Help'. Below the menu bar is a toolbar with buttons for 'Run', 'Next', 'Stop', 'Reset', 'Demo', 'Call for Help', and 'Switch exercise'. The 'Next' button is circled in red. The main window is titled 'Mission: FlowerPot' and contains a code editor with the following code:

```
1 def wakeFlower(c):
2   setBrushColor(c)
3   brushDown()
4   forward(2)
5   backward()
6   left()
7   forward()
8   backward(2)
9   forward()
10  setBrushColor(Color.YELLOW)
11  brushUp()
12  right()
13
14 def line(c):
15   wakeFlower(c1)
16   forward(3)
17   wakeFlower(c2)
18   backward(5)
19 def halfLine(c):
20   forward(2)
21   wakeFlower(c)
22   backward(3)
23
24 def growFlowers():
25   line(Color.RED, Color.CYAN)
26
27   right()
28   forward(2)
29   left()
30   halfLine(Color.ORANGE)
31   right()
32   forward(2)
33   left()
```

The code editor is labeled 'Code area' with a red box. To the right of the code editor is a 'World' grid with a 'Objective' tab. The grid shows a 5x5 area with various colored cells (red, yellow, cyan, green) and a yellow flower icon. A red box labeled 'ongoing execution' is placed over the grid. Below the grid are control buttons: 'forward', 'backward', 'left', 'right', 'mark', and 'Brush Color' (with a dropdown menu). The 'console logs' area at the bottom is empty and labeled 'console logs' with a red box. The status bar at the bottom reads 'Running Brave new world'.

Write your code; Run it: it moves!

# Yet Another IDE for Learners??

- ▶ Exerciser  $\neq$  IDE: Many exercises, Coherent progressions
- ▶ Multi-language: Java/Scala/Python (+ C/javascript/Blockly/Ruby)
- ▶ Multi-lingual: English/French/Brasilian (+ Italian/Chinese)
- ▶ Multi-universe: Several kind of Micro-Worlds

# Yet Another IDE for Learners??

- ▶ Exerciser  $\neq$  IDE: Many exercises, Coherent progressions
- ▶ Multi-language: Java/Scala/Python (+ C/javascript/Blockly/Ruby)
- ▶ Multi-lingual: English/French/Brasilian (+ Italian/Chinese)
- ▶ Multi-universe: Several kind of Micro-Worlds

The collage features several distinct educational environments:

- Top Left:** A grid-based pathfinding exercise with a red path and a small robot icon. Labeled "step 12000".
- Top Middle:** A colorful tree diagram representing a search space or decision tree.
- Top Right:** A grid with a path of letters (A-Z) and a small robot icon.
- Middle Left:** A 3D visualization of a search tree or path, showing a dense structure of nodes.
- Middle Center:** A circular interface with various icons and a central path. Labeled "0 moves".
- Middle Right:** A line graph with multiple colored lines and a dashed line, showing data trends.
- Bottom Left:** A maze with a path and a small robot icon. Labeled "step 12000".
- Bottom Center:** A 3D visualization of a search tree or path, showing a dense structure of nodes.
- Bottom Right:** A control panel with buttons labeled "Main", "Function 1", and "Function 2", and a small graph at the bottom.

length([1, 2, 3])=3  
length([1, 1, 1])=3  
length([1, 2, 1, 3])=4  
length([2, 4, 6, 8, 10])=5  
length(None)=0

# Teaching Material: CS0

## Tactical Programming

- ▶ Solve all initial syntax difficulties
- ▶ Don't get distracted when the algorithms become hairy

## Imperative Kernel (regardless of the language)

- ▶ Instructions & Comments; Conditionals
- ▶ *while* loops; *switch cases*
- ▶ Variables
- ▶ *for* loops and *do/while*
- ▶ Methods, Functions, Parameters
- ▶ Arrays
- ▶ Many application and Summative exercises

200 scenarized exercises (10 to 50+ hours)



# Teaching Recursion in CS1

## Recursion made Simple

- ▶ Split problem; Have a friend solve sub-problem; Gather solutions

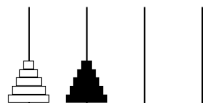
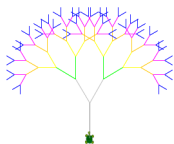
# Teaching Recursion in CS1

## Recursion made Simple

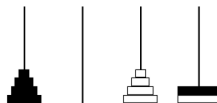
- ▶ Split problem; Have a friend solve sub-problem; Gather solutions

## Supporting Micro-worlds

- ▶ Recursive Lists: `length()`, `isMember()`, etc
- ▶ Turtle Recursion: Trees, Fractals
  - ▶ `subtree()` provided, toward decomposition
- ▶ Hanoi and variations toward decomposition
  - ▶ Linear, Cyclic, Bicolor, Tricolor, etc



Initial Settings



Intermediate Step

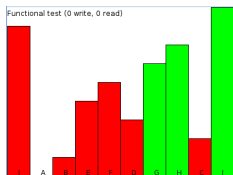


Final Goal

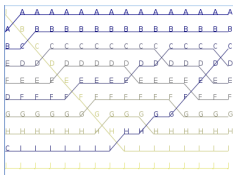
# Teaching Sorting in CS1

Discover: go through a dozen algorithms (enforces #operations)

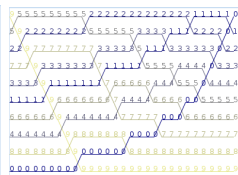
- ▶ State view, and History view to help understanding



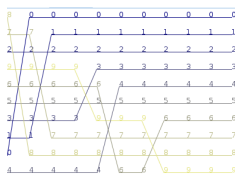
State view



Selection sort

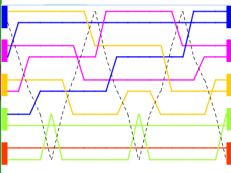
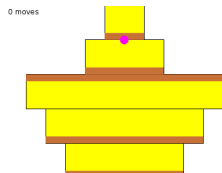


Bubble sort

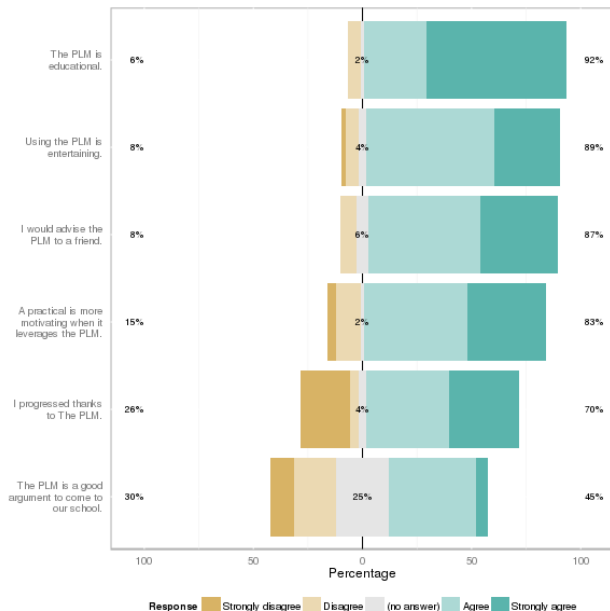


Selection sort

Apply: Dutch Flag, Pancakes or Pebble Motion Problem



# Students' Feedback



# Creating new Material

- ▶ Lesson: Just a graph of exercises (list + dependencies)

# Creating new Material

- ▶ Lesson: Just a graph of exercises (list + dependencies)
- ▶ Exercise: Mission + Initial Worlds + Correcting Entities

The screenshot shows an IDE with a code editor on the left and a game board visualization on the right. The code editor contains the following text:

```
File  
src[] java.util.ArrayList[] myArrayLists[] and myArrayLists[]  
The goal of this exercise is to reproduce the pattern of the first row in the other row as it is a shift of one cell. The objective is to find the biggest difference between the two rows and the other row in terms of the number of cells. The biggest difference between the two rows is the number of cells that you have to read the pattern (or first row) before repeating it. You cannot do otherwise because the same code will be repeated on three different worlds, each of them having a specific pattern.  
The solution is to read the next cell, and go one in addition (going back to read the second cell, but since it is forbidden to use methods, we cannot). The logic is a recursive function (using method) and another, this approach is to be seen as recursive.  
The solution is to store the sequence of values that constitutes the whole pattern in an ArrayList<Integer>[] myArrayLists[] but before we can do so, we should have a list of ArrayList<Integer>.  
src[] java.util.ArrayList[] myArrayLists[]<br>src[] java.util.ArrayList[] myArrayLists[] is an ordered sequence of ArrayList<Integer> objects. Each element of the sequence is identified by its position, and we store a reference to it. ArrayList<Integer>[] is the array name used. Note because arrays are homogeneous in [value], it is possible to represent by using the operator myArrayLists[0].toString() that can contain ArrayList<Integer> or other objects. ArrayList<Integer> is the class name used for the first, because, unlike, it will contain an object variable, but their distinguished counter part Integer, because the ArrayList.toString() is to be treated as good practice to make it an array of integers, not of ArrayList<Integer>[] objects.  
ArrayList<Integer> can store the values of different types, such as Integer, Long, etc.
```

The game board visualization shows a 20x3 grid with a yellow cross in the center. The grid contains various symbols like a bug, a red cross, and a yellow circle. Below the grid is a table with the following data:

Property	Value
World name	Morta
World width	20
World height	3
Selected cell X	0
Selected cell Y	0
Ground color (in 90°/0/90)	90/90/90
Top wall?	N
Left wall?	N
Bugle?	N

```
public class SlugTrackingEntity  
    extends SimpleBuggle {  
  
    // Some additional (hidden) code  
    /* BEGIN TEMPLATE */  
    boolean isFacingTrail() {  
        // Write your code here  
        /* BEGIN SOLUTION */  
        if (isFacingWall())  
            return false;  
        forward();  
        boolean res =  
            getGroundColor().equals(Color.green);  
        backward();  
        return res;  
        /* END SOLUTION */  
    }  
    /* END TEMPLATE */  
}
```

# Creating new Material

- ▶ Lesson: Just a graph of exercises (list + dependencies)
- ▶ Exercise: Mission + Initial Worlds + Correcting Entities



The screenshot shows an IDE with a code editor on the left and a documentation window on the right. The code editor contains Java code for a game world simulation. The documentation window is titled "ArraysLists and Knotting" and contains text explaining the goal of the exercise: to reproduce the pattern of the first row in the other row with a shift. It also includes a small diagram of a 2D grid with a path highlighted.

```
public class SlugTrackingEntity
    extends SimpleBuggle {

    // Some additional (hidden) code
    /* BEGIN TEMPLATE */
    boolean isFacingTrail() {
        // Write your code here
        /* BEGIN SOLUTION */
        if (isFacingWall())
            return false;
        forward();
        boolean res =
            getGroundColor().equals(Color.green);
        backward();
        return res;
        /* END SOLUTION */
    }
    /* END TEMPLATE */
}
```

- ▶ Micro-world: World + View + Entity + ControlPanel
  - ▶ 400 to 1400 lines

# Helping Researchers on CSER

## Do you want to share your data?

The PLM currently saves your usage data anonymously on its servers. That way, you can retrieve your session from any connected computer as long as you sign in.

This anonymous data can also help scientists to understand how people learn programming. With that in mind, we would like to provide your usage data and make it public. [As you can see here](#), no nominative information is published, only the source code written to solve the challenges.

Do you accept to make public your usage data?

Let me decide later

No, don't publish my data

Ok, share my data

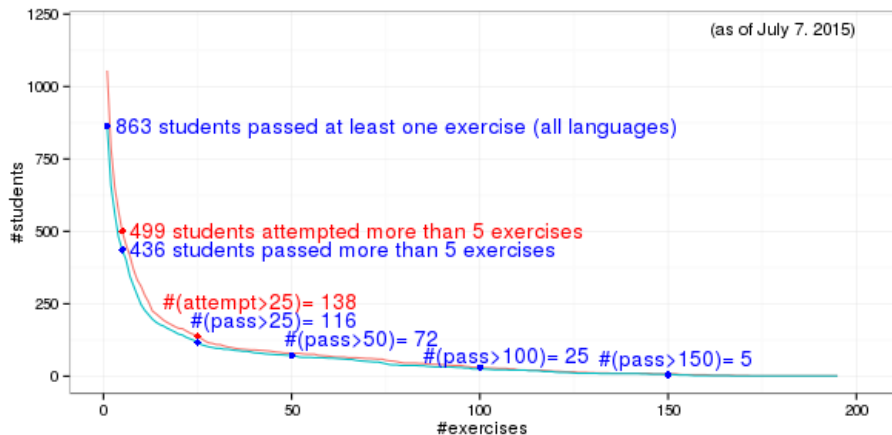
You can later update your preference from your profile's page.

- ▶ Each attempt is saved anonymously, so that it can be rerun
- ▶ 133,636 code submissions (as of yesterday):
  - ▶ 17,480 success; 42,693 compilation errors; 73,463 failures
  - ▶ Scala: 87,514 (66%); Python: 28,066 (21%); Java: 16,669 (13%)



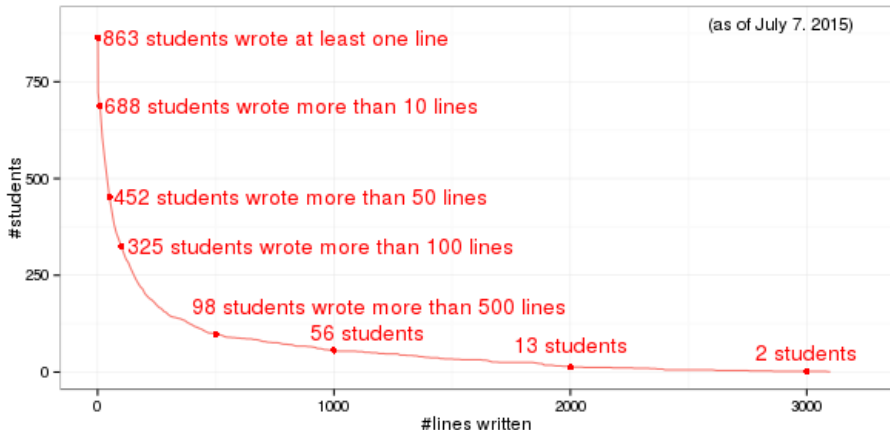
# Exhibiting Longer Learning Path

Cumulative count of students that passed more than X exercises



# Exhibiting Longer Learning Path

Cumulative count of students that wrote more than X lines of code



- ▶ <https://github.com/BuggleInc/PLM-data>
- ▶ We provide a crawler — contact me.

# Current Work on the PLM

## From a Java application to a Web-based service

- ▶ Beta version: <http://plm.telecomnancy.univ-lorraine.fr:9000/>
- ▶ Tests, Distributed Judges and Dockers ongoing
- ▶ Currently at watershed! In production in September

## Teacher Console

- ▶ Dashboard of Class Progress
- ▶ Alerts: Detect stuck students (hard!)

## Personalized Remediation for the MOOCs

- ▶ Clusterize the logic errors (not compile errors)
- ▶ Provide an adapted text to common ones, shown automatically

# Conclusions

## The PLM helps several audiences

- ▶ **Learners:** work at their own pace with a serious «game»
- ▶ **Teachers:** automated+monitoring  $\rightsquigarrow$  more time with students
- ▶ **Authors:** reuse of non-functional code, instrumented feedback
- ▶ **Researchers:** corpus of data

## Future Works:

- ▶ New worlds, new exercises, new languages, polishing
- ▶ Integrated Exercise and Material Editor
- ▶ Integrated Q&A System, community-based programming resource

Join us!

Martin.Quinson@loria.fr