

La Programmer's Learning Machine

- Nécessité d'un outil d'apprentissage de la programmation à TELECOM Nancy
- Création du programme en 2008 par deux enseignants-chercheurs au LORIA
- Proposition de 200 exercices aux élèves

La Programmer's Learning Machine

The screenshot shows the La 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 icons for 'Run', 'Step', 'Stop', 'Reset', 'Demo', 'Call for Help', and 'Switch exercise'. The main window is divided into two panes. The left pane is titled 'Mission' and contains the following text:

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 idea.

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

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

The right pane is titled 'World' and shows a grid representing the world. A small buggle icon is positioned on the grid. A red callout box labeled 'entity' points to the buggle. Below the grid, there is a red callout box labeled 'World view'. At the bottom of the right pane, there are buttons for 'left', 'forward', 'mark', and 'backward', with a red callout box labeled 'Interactive controls' pointing to them. A progress bar at the top of the right pane shows a value of 100.

La Programmer's Learning Machine

The screenshot shows the 'Flower Pot' mission in the La Programmer's Learning Machine. The interface includes a menu bar (File, Session, Language, Help), a toolbar with buttons for Run, Step, Stop, Reset, Demo, Call for Help, and Switch exercise. The 'Demo' button is circled in red. Below the toolbar, the 'Mission' tab is active, displaying the 'Flower Pot' mission details. The mission text describes the goal of reproducing a drawing of a flower pot and provides instructions for writing a `growFlowers()` method. A list of colors is provided: Color.RED, Color.PINK, Color.CYAN, Color.ORANGE, Color.GREEN, and Color.YELLOW. To the right of the mission text is a 'World' view showing a 10x10 grid with a 'Bugle' icon at the bottom left. The 'Objective' tab is selected, showing a target pattern of colored squares. A progress bar at the top of the World view shows the current progress, with the 'Objective' label circled in red. Below the World view are control buttons for 'left', 'forward', 'mark', 'right', and 'backward', along with 'Brush Color' and 'Bugle Color' dropdown menus.

File Session Language Help

Run Step Stop Reset Demo Call for Help Switch exercise

Mission FlowerPot

Flower Pot

Your bugle 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 of the color passed in parameter sounds like a good start, but it is probably not 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

0 50 100 150 200 250 300 350 400 450 500

left forward mark right backward

Brush Color

Bugle Color

La Programmer's Learning Machine

The screenshot displays the La 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. Below the toolbar, the 'Mission' is set to 'FlowerPot', which is also circled in red. The main area is divided into two panes. The left pane contains a code editor with the following code:

```
1 def makeFlower(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  brushDown()
12  right()
13
14 def line(c):
15   makeFlower(c1)
16   forward(3)
17   makeFlower(c2)
18   backward(5)
19 def halfLine(c):
20   forward(2)
21   makeFlower(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 right pane shows a 'World' view with a grid. A slider at the top of the world view ranges from 0 to 500. The grid contains a yellow bug on a cyan square, surrounded by red and green squares. A red box labeled 'ongoing execution' is placed over the grid. Below the world view is a control panel with buttons for 'forward', 'left', 'mark', 'right', and 'backward'. There are also dropdown menus for 'Brush Color' and 'Buggle Color'. A red box labeled 'console logs' is placed at the bottom of the interface. The status bar at the bottom reads 'Running Brave new world'.

La Programmer's Learning Machine

Avantages :

- Grande diversité des exercices
- Apprentissage ludique, attrayant et interactif
- Travail en autonomie, à son propre rythme
- Motivation de l'élève grâce au retour rapide sur son travail
- Système évolutif
- Exercices pouvant être résolus dans différents langages

Ma mission

- Besoin de créer des exercices plus simplement
- Réalisation d'un éditeur d'exercices comprenant :
 - Création des mondes
 - Création de la mission
 - Création du code de la solution
 - Chargement et sauvegarde d'exercices

Mes outils



L'éditeur de mondes

- Création de mondes de type BuggleWorld
- Commandes pour ajouter / supprimer des éléments, modifier la grille
- Tableau des propriétés du monde
- Gestion de plusieurs mondes

L'éditeur de mondes

BuggleQuest - MapEditor

File



The main editing area shows a 23x18 grid with a maze structure. The maze is defined by blue walls and contains several brown circular obstacles. A red square highlights the top-left cell (0,0), and a yellow square highlights a cell at the top-right of the maze.

Property	Value
World name	Deep Forest
World width	23
World height	18
Selected cell X	0
Selected cell Y	0
Ground color (name or r/g/b)	white
Top wall?	N
Left wall?	N
Baggle?	N

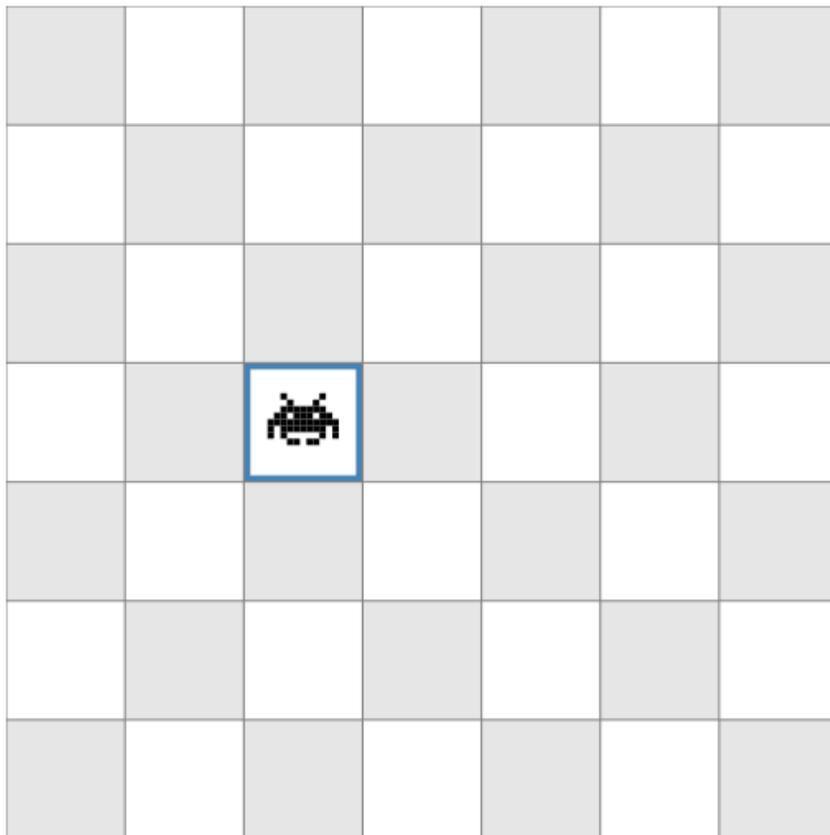
L'éditeur de mondes

Select cell Top wall ▼ Baggle Buggle ▼ Color Text Add line ▼

Worlds ▼

New world ▼

Add World Delete World



Property	Value
World name	<input type="text" value="New world"/>
World width	<input type="text" value="7"/>
World height	<input type="text" value="7"/>
Cell X	2
Cell Y	3
Message	<input type="text"/>
Color	<input type="text" value="white"/>
Top wall?	<input type="checkbox"/>
Left wall?	<input type="checkbox"/>
Baggle?	<input type="checkbox"/>

Property	Value
Buggle name	<input type="text" value="buggle1"/>
Buggle direction	<input style="border: none; border-bottom: 1px solid #ccc; background-color: #f0f0f0; padding: 2px 5px;" type="text" value="North"/>
Buggle color	<input type="text" value="black"/>

L'éditeur de missions

- Rédaction de la mission
- Affichage en temps réel du rendu final
- Choix du langage de programmation pour le rendu

L'éditeur de missions

The screenshot shows a window titled "PLM - Mission Editor" with a "File" menu. The left pane contains a code editor with HTML and code snippets. The right pane has a language selection bar (All, C, Java, Scala, Python) and an "Instructions" section with text and a code example.

Code Editor (Left Pane):

```
<p>For now,
we'll simply go for the buggle instructions. There is a met
interactive control panel. To achieve the same effect tha
(making the buggle moving one step forward), you need
editor: </p>
<pre>[!java|scala|python]forward()[!java];[!][!][!c]step
<p>Likewise, to achieve the same effect than the <b>[!
epBackward[!]</b>, <b>left</b>
and <b>right</b> buttons, you need to use respectively
<pre>
backward()[!java|c];[!|]
left()[!java|c];[!|]
right()[!java|c];[!|]
</pre>

<p>The <b>mark</b> button is a bit parti
methods: the first one moves the pen up while the second
<pre>
brushUp()[!java|c];[!|]
brushDown()[!java|c];[!|]
</pre>
<p>The buggle offers other methods, that are presente
this world" menu and will be introduced on need.</p>

<h3>Exercise goal</h3><a name="Objectives"/>
Our second program will be a bit more complicated, but r
your buggle is simply to draw a house (a box), and hide i
objective world to see exactly what this means.

<p>Use your brush to mark the ground as you walk. Sta
<code>brushDown()</code>, and stop it with <code>b
```

All C Java Scala Python

Instructions

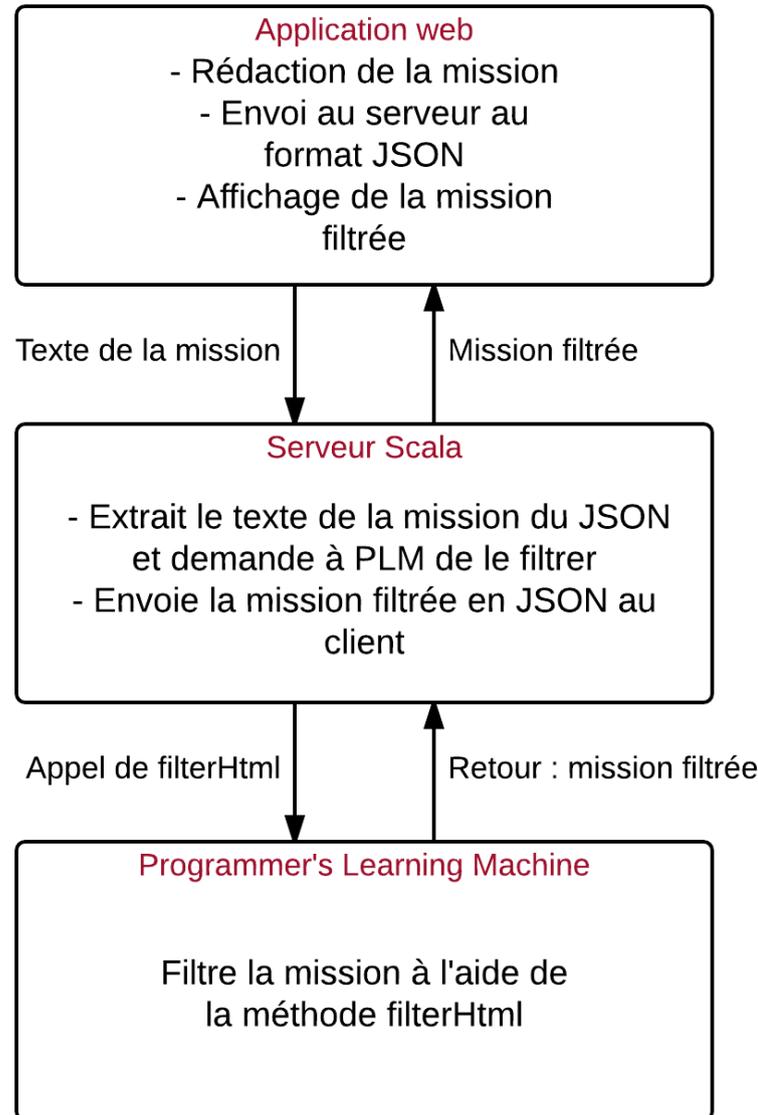
Congratulations! You just wrote your first program! You got the idea now: programming is nothing more than giving simple instructions to the computer that blindly apply them. The main difficulty is to explain stuff to something as stupid as a computer...

Programs are mainly suites of method calls, which are no more than a list of simple orders given to the machine. It is very similar to a recipe stating *Melt the chocolate pieces, add sugar, cool the mix and serve*. In your programs, such built instructions are called functions or methods, and you should add parenthesis to invoke them:

```
nameOfTheMethod()
```

JavaPythonScalaC wants to have the instructions separated by semi-columns (;) or by new lines. The previous example would thus be written in the following way (you can also add semi-columns at the end of the lines, but this is not mandatory).

L'éditeur de missions



L'éditeur de missions

All C java Scala Python

<h2>La buggle Petite Poucette</h2>

<p>Votre buggle est perdue dans un étrange labyrinthe, et elle a besoin de vous pour trouver la sortie (représentée par les cases orange). Vous ne pouvez pas lui donner son chemin tout simplement avec quelque chose comme `<code>droite();avance();avance();</code>` parce qu'il faut secourir deux buggles à la fois, perdues dans des labyrinthes similaires mais différents. Vous pouvez passer à l'autre monde en cliquant sur le menu défilant au dessus de l'endroit où est dessiné le monde. C'est là où il est écrit "Deep Forest" pour l'instant (forêt profonde), et si vous passez à "Deeper Forest" (forêt encore plus profonde), vous verrez l'autre monde.</p>

<p>La bonne nouvelle est que le chemin vers la sortie est en quelque sorte écrit au sol. Ces mondes sont composés de plusieurs corridors, avec des baggles par terre. À chaque embranchement, il faut prendre à gauche si le corridor qu'on vient de parcourir contient 3 baggles ou plus, ou à droite s'il contient 2 baggles ou moins.</p>

<p>Vous devez compter exactement 5 cases par couloir. Les cases aux intersections doivent être comptées comme les dernières de leur couloir, pas comme les premières après avoir tourné. </p>

<p>La forme générale de votre code doit donc être quelque chose comme «tant que je n'ai pas trouvé la sortie, prendre le prochain couloir pour décider s'il

La buggle Petite Poucette

Votre buggle est perdue dans un étrange labyrinthe, et elle a besoin de vous pour trouver la sortie (représentée par les cases orange). Vous ne pouvez pas lui donner son chemin tout simplement avec quelque chose comme `droite();avance();avance();` parce qu'il faut secourir deux buggles à la fois, perdues dans des labyrinthes similaires mais différents. Vous pouvez passer à l'autre monde en cliquant sur le menu défilant au dessus de l'endroit où est dessiné le monde. C'est là où il est écrit "Deep Forest" pour l'instant (forêt profonde), et si vous passez à "Deeper Forest" (forêt encore plus profonde), vous verrez l'autre monde.

La bonne nouvelle est que le chemin vers la sortie est en quelque sorte écrit au sol. Ces mondes sont composés de plusieurs corridors, avec des baggles par terre. À chaque embranchement, il faut prendre à gauche si le corridor qu'on vient de parcourir contient 3 baggles ou plus, ou à droite s'il contient 2 baggles ou moins.

Vous devez compter exactement 5 cases par couloir. Les cases aux intersections doivent être comptées comme les dernières de leur couloir, pas comme les premières après avoir tourné.

La forme générale de votre code doit donc être quelque chose comme «tant que je n'ai pas

L'éditeur de solutions

- Rédaction de la solution dans les différents langages de programmation
- Visualisation du résultat sur le monde

L'éditeur de solutions

Solution code

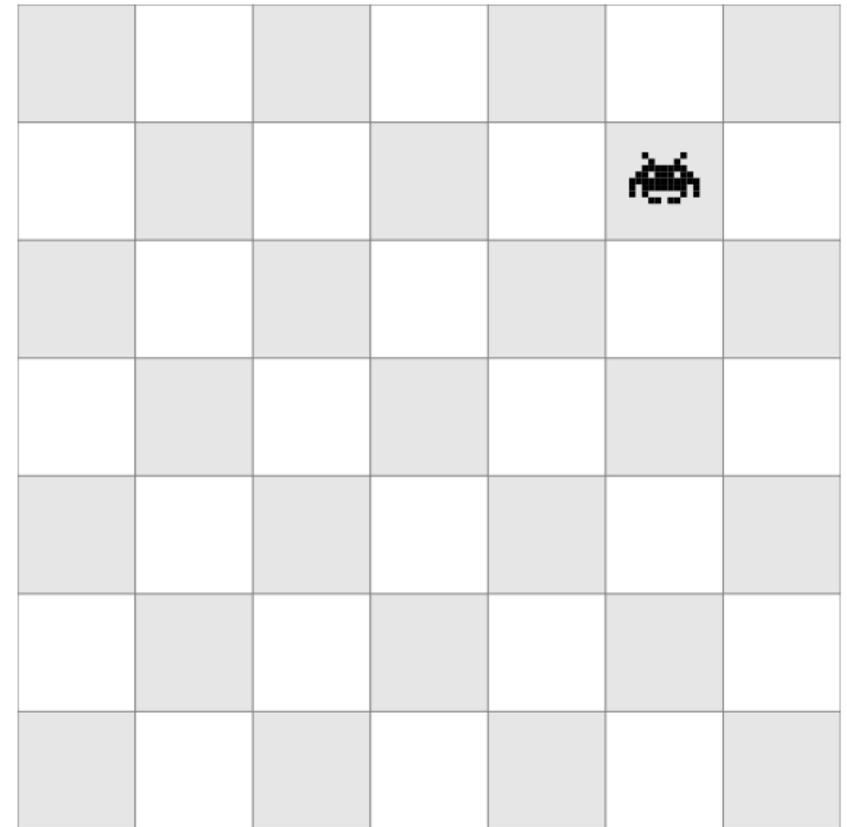
API

Java

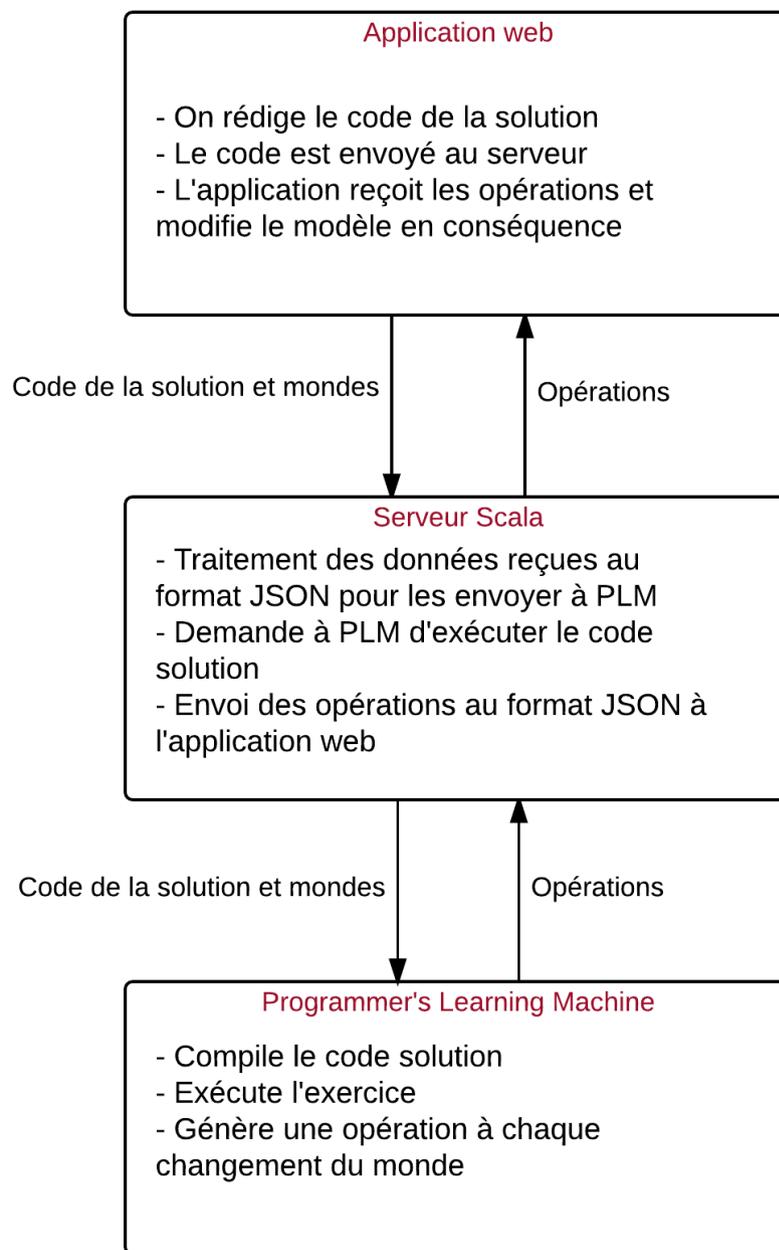
Training World

Exécuter

```
0 /* BEGIN TEMPLATE */
1 /* BEGIN SOLUTION */
2 brushDown();
3 left();
4 forward();
5 forward();
6 forward();
7 forward();
8 left();
9 forward();
10 forward();
11 forward();
12 forward();
13 left();
14 forward();
15 forward();
16 forward();
17 forward();
18 left();
19 forward();
20 forward();
21 left();
22 forward();
```



L'éditeur de solutions



Chargement et sauvegarde

- Chargement d'un exercice existant
- Sauvegarde des données d'un exercice dans un dossier sur le serveur
 - Classe principale de l'exercice
 - Mondes dans des fichiers .map
 - Mission dans un fichier .html
 - Codes solutions dans des fichiers sources

Améliorations possibles

- Compatibilité avec certains exercices
- Meilleur système de sauvegarde
- Galerie d'exercices

Conclusion

- Contribution à un projet intéressant et motivant
- Acquisition de nouvelles connaissances
- Découverte du monde professionnel