# PROGRAMMER'S LEARNING MACHINE

MATTHIEU NICOLAS

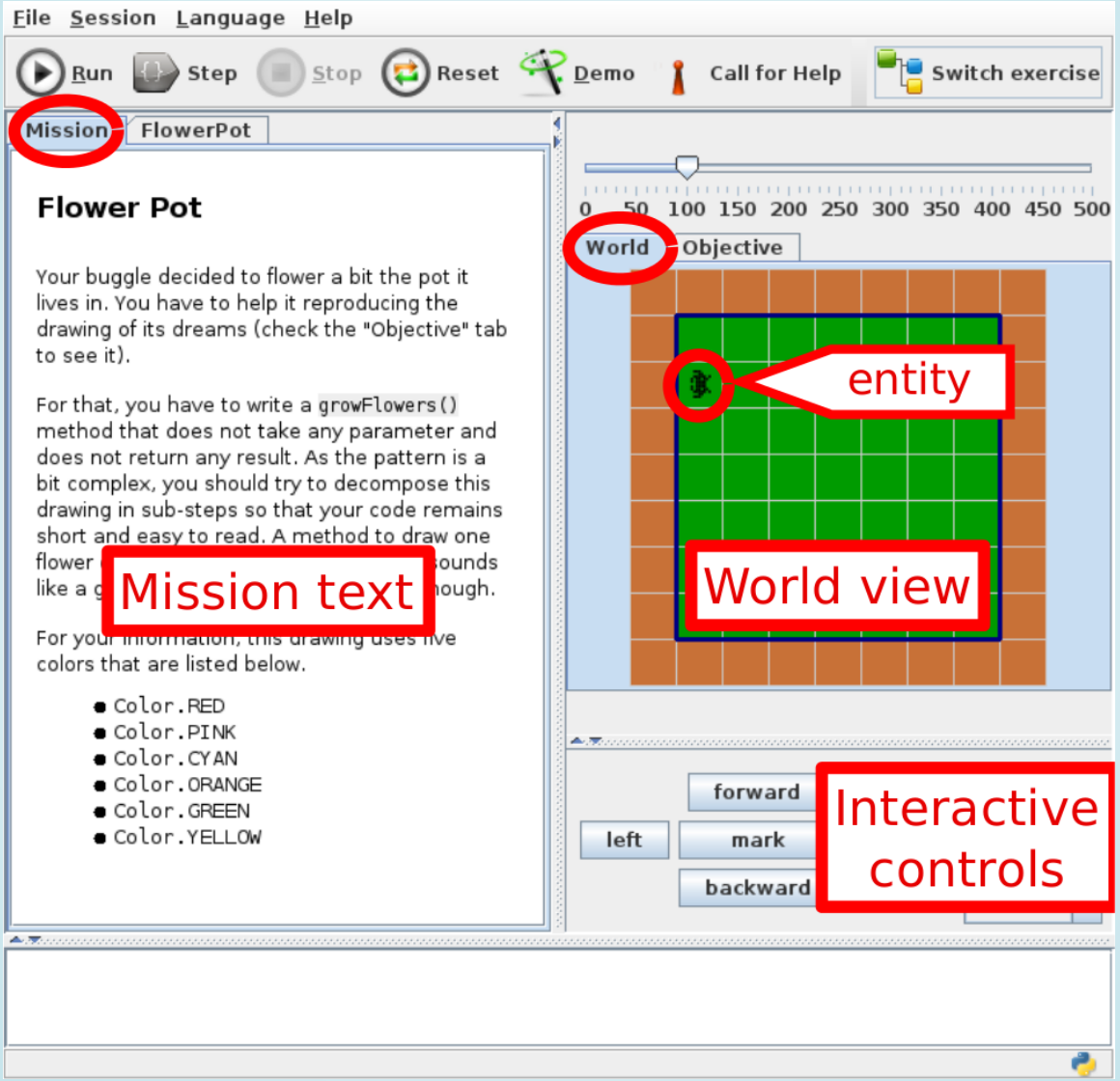
IJD SEMINAR - MARCH 03, 2015

# WHAT IS PLM

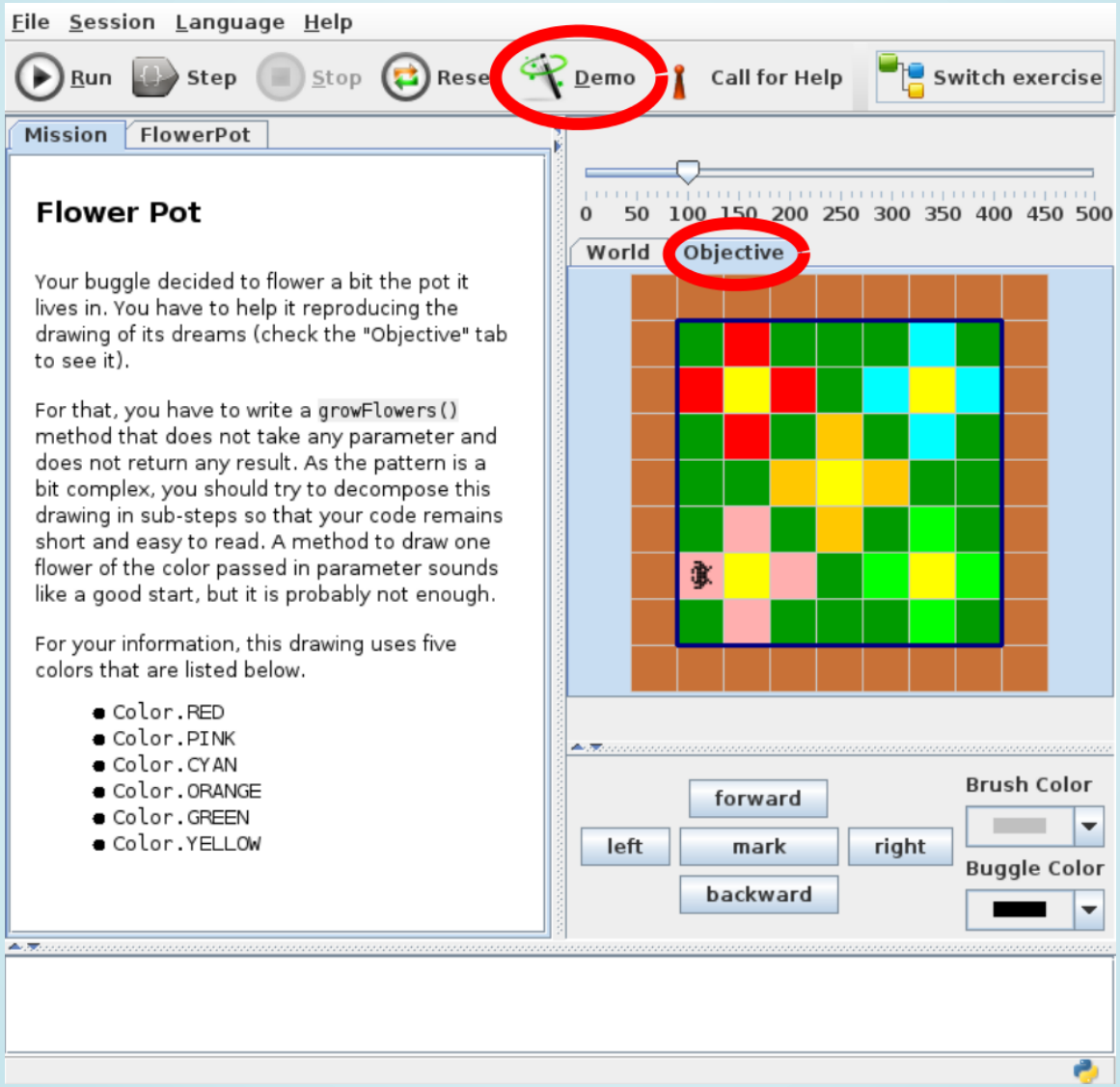
- Software to learn programming
- Allows students to progress at their own speed...
- ... while the teacher helps the ones having trouble
- Used at *TELECOM Nancy* since 2008

# STATE BEFORE ADT

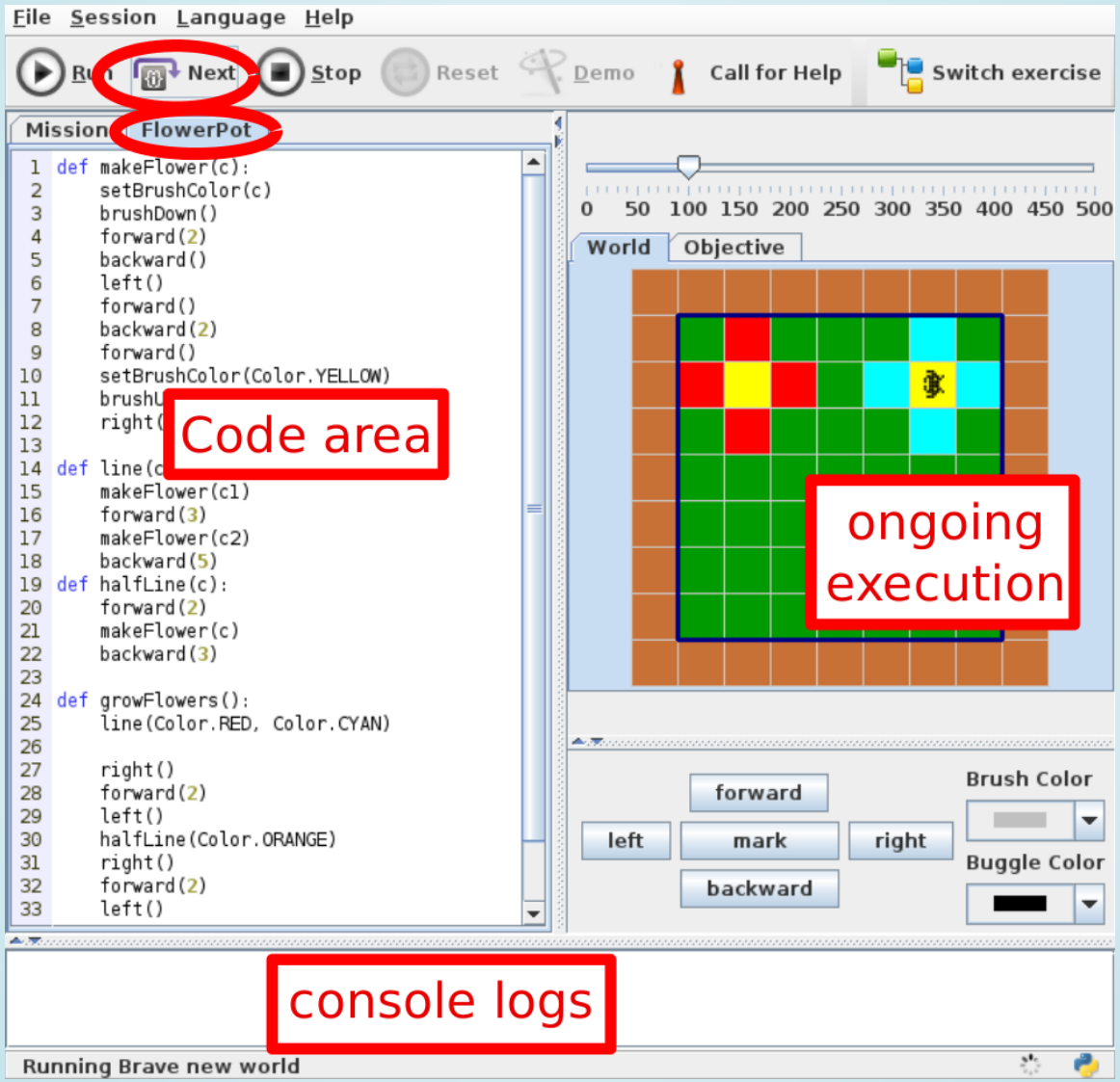
# OVERVIEW


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
Mission text
colors that are listed below.
Color.RED
Color.PINK
Color.CYAN
Color.ORANGE
Color.GREEN
Color.YELLOW
0 50 100 150 200 250 300 350 400 450 500
World
Objective
entity
World view
forward
left
mark
backward
Interactive controls

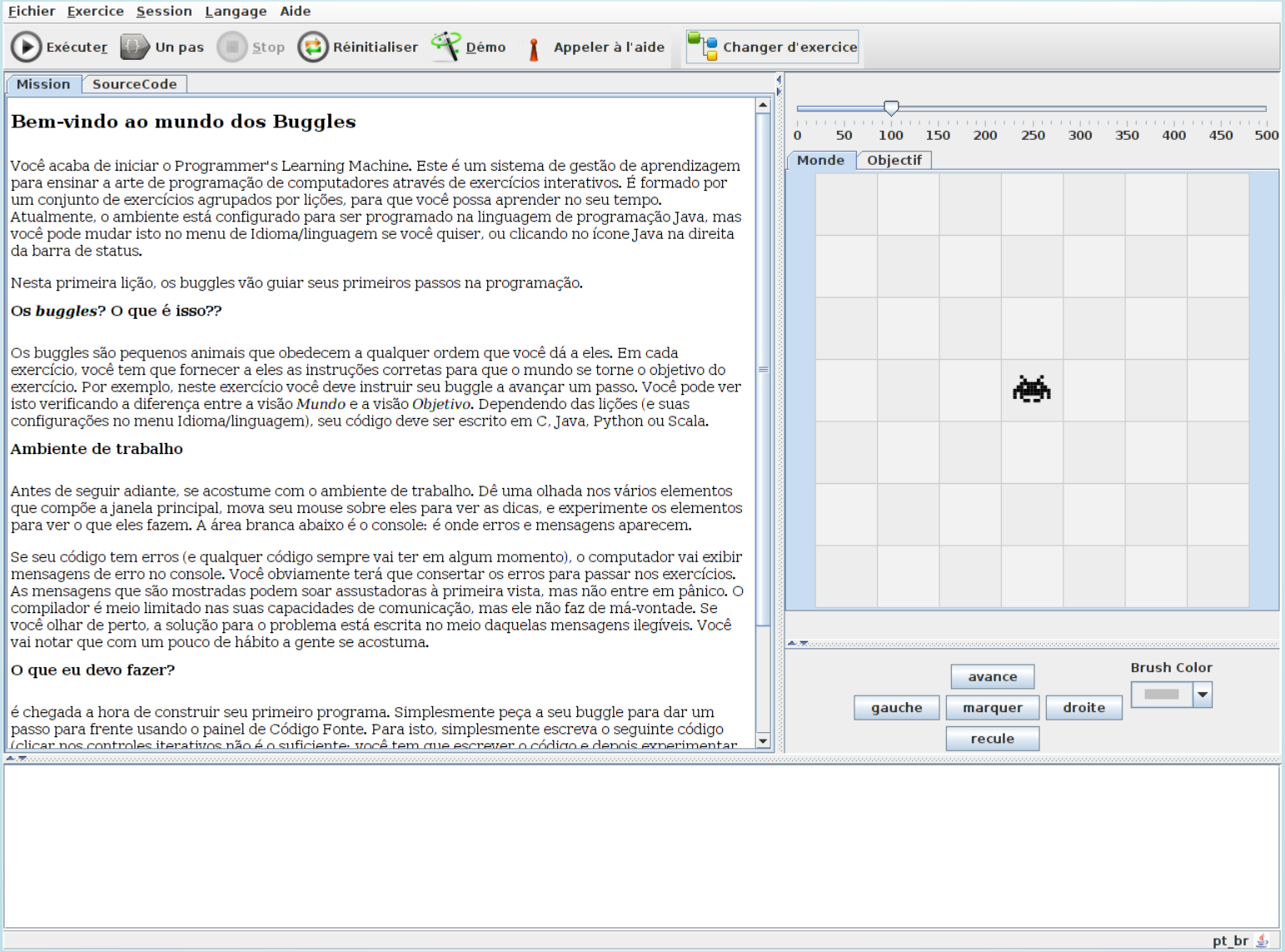
# OVERVIEW


File Session Language Help
Run
Step
Stop
Rese
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 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
0 50 100 150 200 250 300 350 400 450 500
World
Objective
forward
left
mark
right
backward
Brush Color
Buggle Color

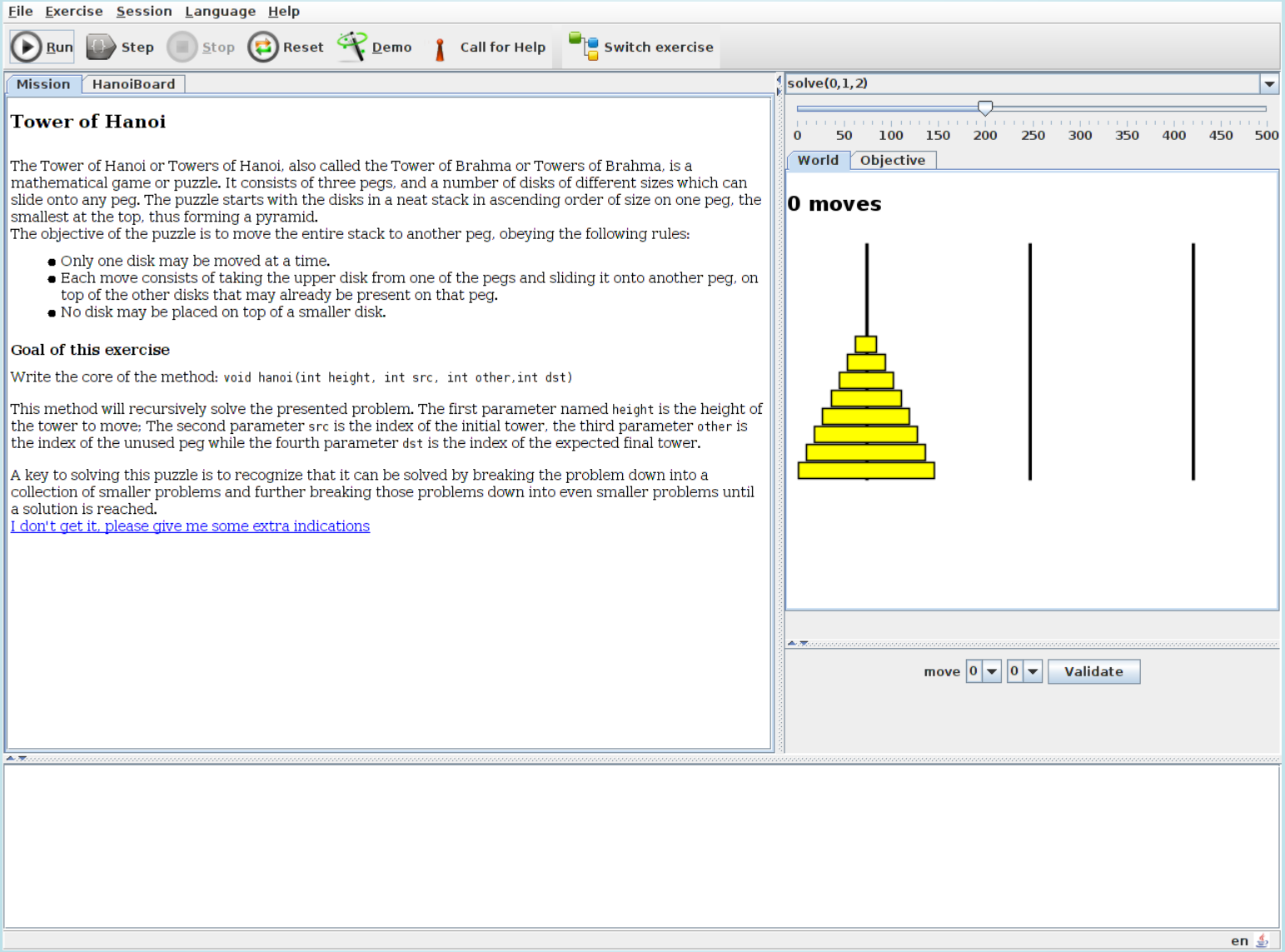
# OVERVIEW


File Session Language Help
Run
Next
Stop
Reset
Demo
Call for Help
Switch exercise
Mission
FlowerPot
def makeFlower(c):
setBrushColor(c)
brushDown()
forward(2)
backward()
left()
forward()
backward(2)
forward()
setBrushColor(Color.YELLOW)
def line(c
makeFlower(c1)
forward(3)
makeFlower(c2)
backward(5)
def halfLine(c):
forward(2)
makeFlower(c)
backward(3)
def growFlowers():
line(Color.RED, Color.CYAN)
right()
forward(2)
left()
halfLine(Color.ORANGE)
right()
forward(2)
left()
Code area
0 50 100 150 200 250 300 350 400 450 500
World
Objective
ongoing execution
forward
left
mark
right
backward
Brush Color
Buggle Color
console logs
Running Brave new world

# 200 EXERCISES



# 200 EXERCISES



# 200 EXERCISES

Fichier Exercice Session Langage Aide

Exécuter | Un pas | Stop | Réinitialiser | Démo | Appeler à l'aide | Changer d'exercice

Mission | Lander

## Se poser pour les nuls

Bravo ! Vous avez gagné dans une boîte de céréales un voyage (aller simple) pour Mars. La NASA vous a même fourni un module d'excursion lunaire pour l'occasion, et c'est donc avec ce véhicule que vous voyagerez.

Avant le décollage, vous devez vous entraîner sur simulateur aux bases de l'alunissage. Si tout se passe comme prévu, vous ne devriez pas avoir à voler en mode manuel, mais mieux vaut prévenir que guérir, comme ils disent.

Votre mission est de **poser le module avec une vitesse verticale inférieure à 10 m/s.** Il y a deux façons d'influer sur la trajectoire du module : en modifiant son angle, ou en réglant la poussée de son moteur. Dans cette simulation, nous ne nous intéressons qu'à la poussée, exprimée en m/s².

Vous devez écrire le code qui sera appellé tous les dixièmes de seconde pendant la simulation. Assurez vous que votre code ne dure pas trop longtemps, et évitez en particulier les boucles `while`. Il faut juste ajuster la poussée du moteur en utilisant la fonction `void setPousseeDesiree(int poussee)` où `poussee` est un entier entre 0 et 4 (représentant une poussée entre 0 m/s² et 4 m/s²). Vous ne pouvez incrémenter ou décrémenter la poussée que de 1 entre deux pas de la simulation. Ainsi, si la poussée du moteur est actuellement de 2 et que vous demandez 4, vous n'aurez que 3 à la prochaine étape. Si vous demandez une valeur inférieure à 0 (ou supérieure à 4), tout se passera comme si vous aviez demandé 0 (ou 4).

Afin de prendre des décisions informées, vous pouvez demander des informations sur l'état actuel du module. Dans cette simulation, vous ne serez probablement intéressé que par sa position verticale (accessible grâce à `double getY()`) et sa vitesse verticale (accessible grâce à `double getVitesseY()`). Vous pouvez également demander la poussée actuelle du moteur (avec `int getPoussee()`) si vous avez oublié ce que vous avez demandé. Rappelez vous que la gravité martienne est de 3.711 m/s², cela peut vous être utile.

Une dernière chose : le module est plutôt petit, et vous n'avez pas tant de carburant que cela. À chaque pas de simulation, il consomme autant d'unité de carburant que la poussée actuelle du moteur. Si vous tombez en panne sèche, vous serez en chute libre alors attention ! Vous pouvez consulter le niveau du réservoir avec `int getFuel()`.

Bon courage !

0 50 100 150 200 250 300 350 400 450 500

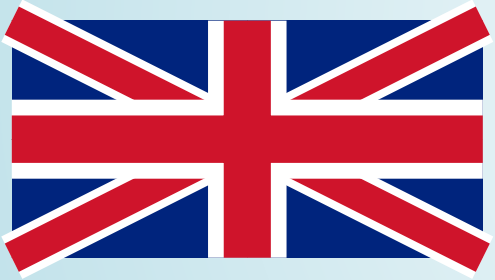
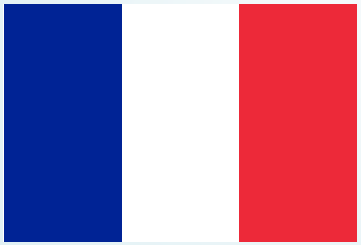
Monde | Objectif

x: 1200,00
y: 700,00
speed x: 0,00
speed y: 0,00
angle: 0,00°
thrust: 0
fuel: 3000

fr

# PROGRAMMING LANGUAGES

# LANGUAGES

# USER TRACKING

- Keep the students' sessions
- Track the students' progress

# USER TRACKING

- **git** used in order to version the student's code
- Local repository and anonymous branch
- Data pushed to a **GitHub** repository

# USER TRACKING

- User's actions stored as commits

```
{
    kind:"executed",
    lang:"Java",
    exo:"welcome.lessons.welcome.instructions.Instructions",
    passedtests:"1",
    totaltests:"1",
    outcome:"pass"
}
```

# ADT'S GOALS

# IMPROVE THE SOFTWARE'S QUALITY

# ATTRACT STUDENTS

- More teaching content
- Webification
- Gamification

# AND TEACHERS

- Keep track of the students' progress
- Adapt content to their needs
- Able to add their own exercises

# AND RESEARCHERS

- To an experimental teaching platform
- How to detect students having difficulties?
- What are the most common errors?

# WORK DONE

# FIRST STEPS

- Back-to-school season
  - Got the students' feedback...
  - ... and 170 bug reports
- Solved minor issues

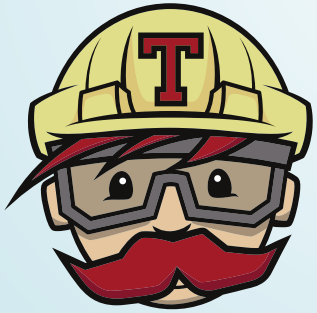
# DEBUG USER TRACKING

- Lost data
- Fixed and refactored the code

# UNIT TESTING

- Need to ensure the critical parts
  - **git**
  - exercises' solutions
  - lessons

# CONTINUOUS INTEGRATION



**Travis CI**

# CONTINUOUS INTEGRATION

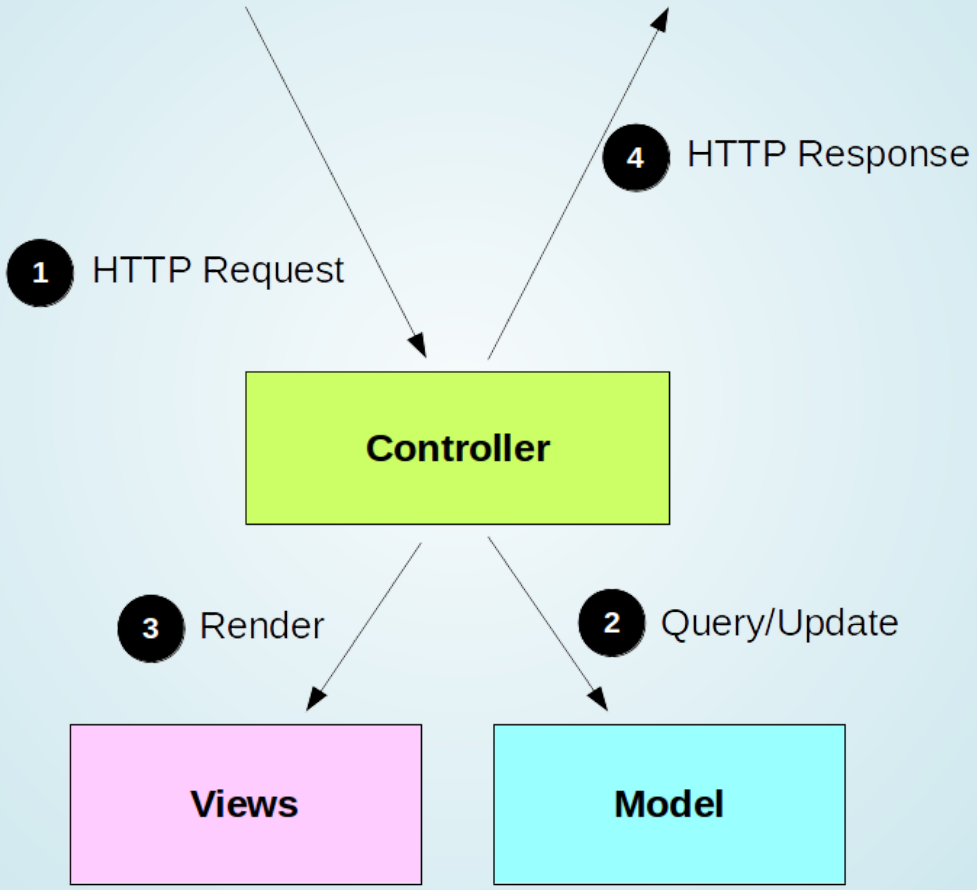
- Execute tests automatically when commits are pushed
- Notify us as soon as a build failed

# TO A WEB APP

# PLAY FRAMEWORK

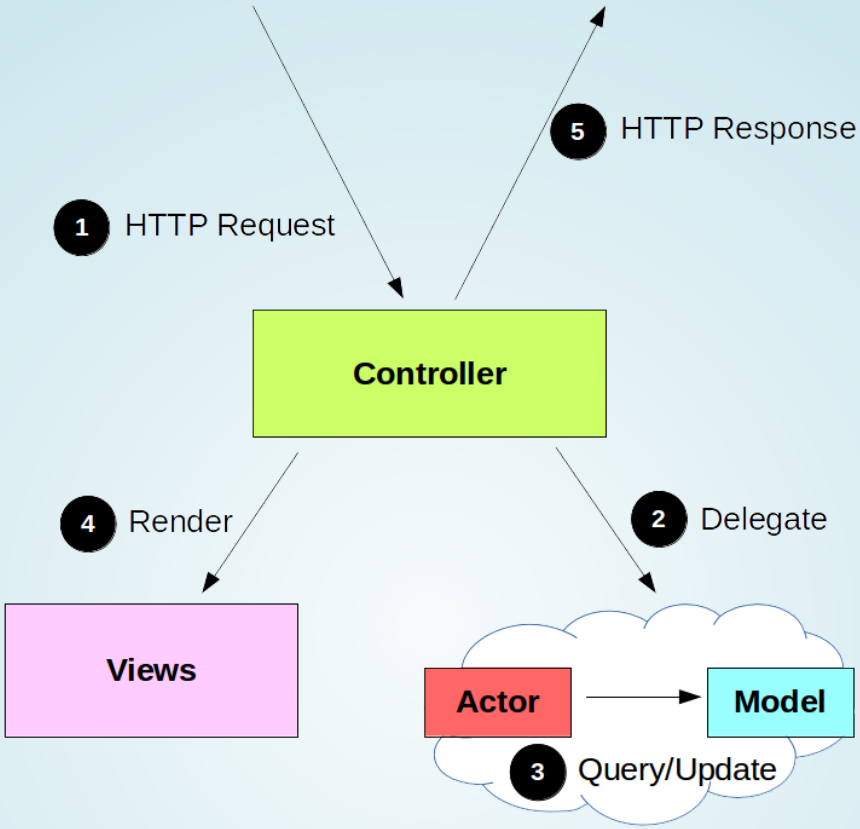
- Java and Scala web application framework
- Allow to set up application server

# MVC PATTERN


4 HTTP Response
1 HTTP Request
Controller
3 Render
2 Query/Update
Views
Model

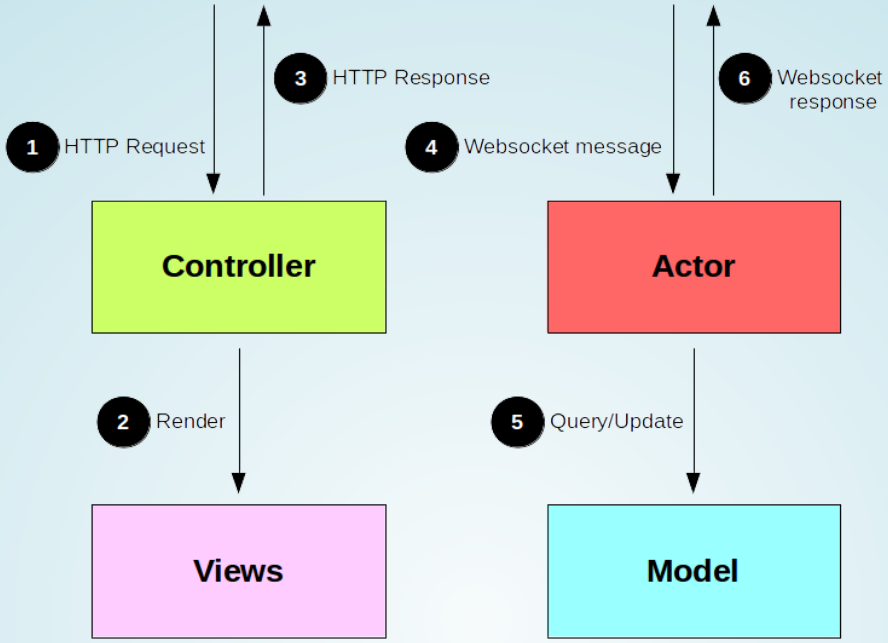
# ACTORS

- Concurrent and scalable
- *'Let it crash'*


5 HTTP Response
1 HTTP Request
Controller
4 Render
2 Delegate
Views
Actor
Model
3 Query/Update

# WORKING WITH WEBSOCKETS

- Controller only needed to render views
- One actor per websocket


1 HTTP Request
3 HTTP Response
4 Websocket message
6 Websocket response
Controller
Actor
2 Render
5 Query/Update
Views
Model

# REFACTOR PLM

- To be used as a library
- Remove the current UI
- Keep track of the world's evolution

# REFACTOR PLM

- Has to update the server's and the client's world model
- Each time the world is modified...

```
setBugglePosition(newX, newY)
```

- ... the corresponding operation is generated

```
//Operation stored then send to the client
moveBuggleOperation(oldX, oldY, newX, newY)
```

# CLIENT-SIDE

- One-page application built with **AngularJS**
- UI made with **Foundation**

# CLIENT-SIDE

# RESULTS

- Main functionalities implemented
- First universe adapted
- Only as a local server

# NEXT STEPS

- Convert other universes
- Embed debugging tools
- Support **C** language
- To a web server

# THANKS FOR LISTENING

Do you have any questions?