

IUT Nancy-Charlemagne

Université de Lorraine

2 ter Boulevard Charlemagne

54052 Nancy Cedex

Dépt. Informatique

**Langage visuel pour un exerciseur**

Rapport de stage DUT informatique

LORIA

Baptiste MOUNIER

2015

**Langage visuel pour un exerciseur**

Gérald OSTER

Isabelle DEBLED-RENNESSON

IUT Nancy-Charlemagne

Université de Lorraine

2 ter Boulevard Charlemagne

54052 Nancy Cedex

Dépt. Informatique

Rapport de stage DUT informatique

LORIA

Campus Scientifique

BP 239

54506 Vandœuvre-lès-Nancy

Baptiste MOUNIER

# 

# Remerciements

Il me semble judicieux avant de débuter ce rapport de remercier toutes celles et ceux qui m’ont permis de le réaliser ces conditions.

Je commencerai par Matthieu Nicolas, Gérald Oster et Martin Quinson qui m’ont accepté au sein de leur projet et qui m’ont accompagné dans la réalisation de ma contribution.

Egalement Mme. Debled-Rennesson, ma marraine de stage qui a effectué le suivi et l’encadrement de mon stage.

De même que toute l’équipe COAST qui m’a accueilli dans leurs locaux avec la joie et la bonne humeur, éléments essentiels d’un travail efficace au quotidien. Notamment Claudia-Lavina [Oster/Ignat]? Et Luc André. Et par extension le LORIA dans son ensemble.

Je n’oublierai pas l’IUT et ses enseignants qui m’ont apporté tout au long de ces deux années les capacités pour réaliser ce genre de projet.

Table des matières

[Remerciements 2](#_Toc421943226)

[Introduction 4](#_Toc421943227)

[Sujet 4](#_Toc421943228)

[Mise en œuvre 4](#_Toc421943229)

[LORIA 6](#_Toc421943230)

[Laboratoire 6](#_Toc421943231)

[Equipe COAST 6](#_Toc421943232)

[Contexte 7](#_Toc421943233)

[Etat du projet 7](#_Toc421943234)

[Blockly 7](#_Toc421943235)

[Outils, technologies et méthodes 8](#_Toc421943236)

[Développement 9](#_Toc421943237)

[Découverte 9](#_Toc421943238)

[Insertion 9](#_Toc421943239)

[Sélection 9](#_Toc421943240)

[Affichage 10](#_Toc421943241)

[Génération 11](#_Toc421943242)

[Toolbox 11](#_Toc421943243)

[Bloc 11](#_Toc421943244)

[Liaison 13](#_Toc421943245)

[Stockage 14](#_Toc421943246)

[Traduction 16](#_Toc421943247)

[Conclusion 18](#_Toc421943248)

[Bibliographie 19](#_Toc421943249)

[Sites web 19](#_Toc421943250)

[Rapport 19](#_Toc421943251)

# Introduction

## Sujet

Le stage consiste à la réalisation de l’ajout de Blockly au sein de l’application web de PLM (Programmer’s Learning Machine). Utilisant les langages Java, Python, Scala et C, Blockly ajoute une couche visuelle afin de facilité la première approche vis-à-vis de la programmation.

Blockly est elle-même une application réalisée par Google sous licence Apache 2.0. C’est une librairie de développement pour logiciels et applications d’apprentissage de la programmation. Son utilisation est très visuel est sous forme de « drag and drop » de blocs représentant des instructions, comme une affectation d’un entier à une variable, dans un espace de travail.

PLM est une application, récemment portée sur le web, d’apprentissage de la programmation développée par l’équipe COAST du LORIA sous licence publique générale GNU v3.0. L’application est actuellement utilisée en école d’ingénieur, Télécom Nancy, pour une progression rapide des étudiants découvrant la programmation. Dans la suite de ce rapport j’utiliserai « webPLM » comme terme générique pour désigner le côté web de l’application et « PLM » pour désigner le côté logiciel existant déjà avant le portage web PLM.

## Mise en œuvre

Mon travail s’est effectué en 5 grands axes que je développerais dans cet ordre dans ce dossier (réf : Annexe 1 – Diagramme de Gantt).

Avant tout, j’ai dû découvrir un projet en cours de réalisation et effectuer une légère remise à niveau sur les technologies employées dans sa conception.

Pour ma contribution, il m’a d’abord fallu insérer Blockly dans le projet, tant au niveau de PLM que webPLM. Simplement pour pouvoir dans un premier temps sélectionner Blockly et avoir une base sur laquelle poursuivre l’implémentation.

L’étape suivante a été de mettre en place la barre d’outils proposant les blocs utilisables. C’est à partir de cette étape que j’ai commencé à avoir une réelle interaction entre l’utilisateur et l’application via Blockly. Cette partie regroupe également le fonctionnement des blocs.

Puis la gestion du stockage du travail réalisé. Utilisé pour la simple sauvegarde de l’avancement dans les exercices, le stockage est aussi indispensable pour réaliser le suivi des étudiants par les enseignants.

Et enfin de la gestion de la traduction, de l’anglais qui est le statut « par défaut » de l’application au français. Le projet a une portée supérieure à la simple application en milieu scolaire, même dans ce cas, le portage en anglais et en français de l’application est un véritable plus d’accessibilité.

Dans tout ce rapport je suivrais un exemple, celui de l’exercice « Moria » dans l’univers « Welcome ».

# LORIA

## Laboratoire

Le LORIA est le Laboratoire Lorrain de Recherche en Informatique et ses Applications. Créé en 1997, il est membre, avec les deux autres principaux laboratoires de recherche en mathématiques et STIC (Science et Technologie de l’Information et de la Communication) de Lorraine, de la Fédération Charles Hermite. C’est une Unité Mixte de Recherche (UMR 7503) entre le CNRS (Centre National de la Recherche Scientifique), l’Université de Lorraine et Inria (Institut National de Recherche en Informatique et en Automatique).



Figure 1 : Logo LORIA



Figure 2 : Logo Inria



Figure 3 : Logo CNRS

Le laboratoire est structuré autour d’une 30aine d’équipes répartis en 5 départements chacun représentant depuis 2010 une thématique (réf : Annexe 1 - Organigramme 2014 LORIA) pour un total de plus de 450 personnes.

## Equipe COAST

Le stage s’est déroulé au sein de l’équipe COAST du LORIA et plus particulièrement en collaboration direct avec Gérald Oster, enseignant chercheur, et Matthieu Nicolas, ingénieur interne. Egalement avec deux autres stagiaires travaillant également sur PLM mais sur des problématiques différentes, Benjamin Thirion et Theodore Lambolez.

L’équipe, dirigée par François Charoy, fait partie du 3ème département « Réseaux, système et services » et travail majoritairement sur la conception de service de partage d’objets, de communication, de gestion de tâches, de maintien d’une conscience de groupe, d’aide à la prise de décisions. Notamment des applications de co-conception et/ou de co-ingénierie en Génie Logiciel, Architecture, Formation-Apprentissage.

# Contexte

## Etat du projet

PLM est un programme débuté en 2008 et en amélioration constante avec une mise à jour régulière suite aux retours des utilisateurs. L’application est développée en Java. Elle permet la réalisation en différents langages (Java, Python, C, Scala) de plus de 190 exercices répartis en 14 univers afin d’apprendre la programmation à son rythme. Il est également possible de choisir entre l’anglais, le français, l’italien, et le portugais.

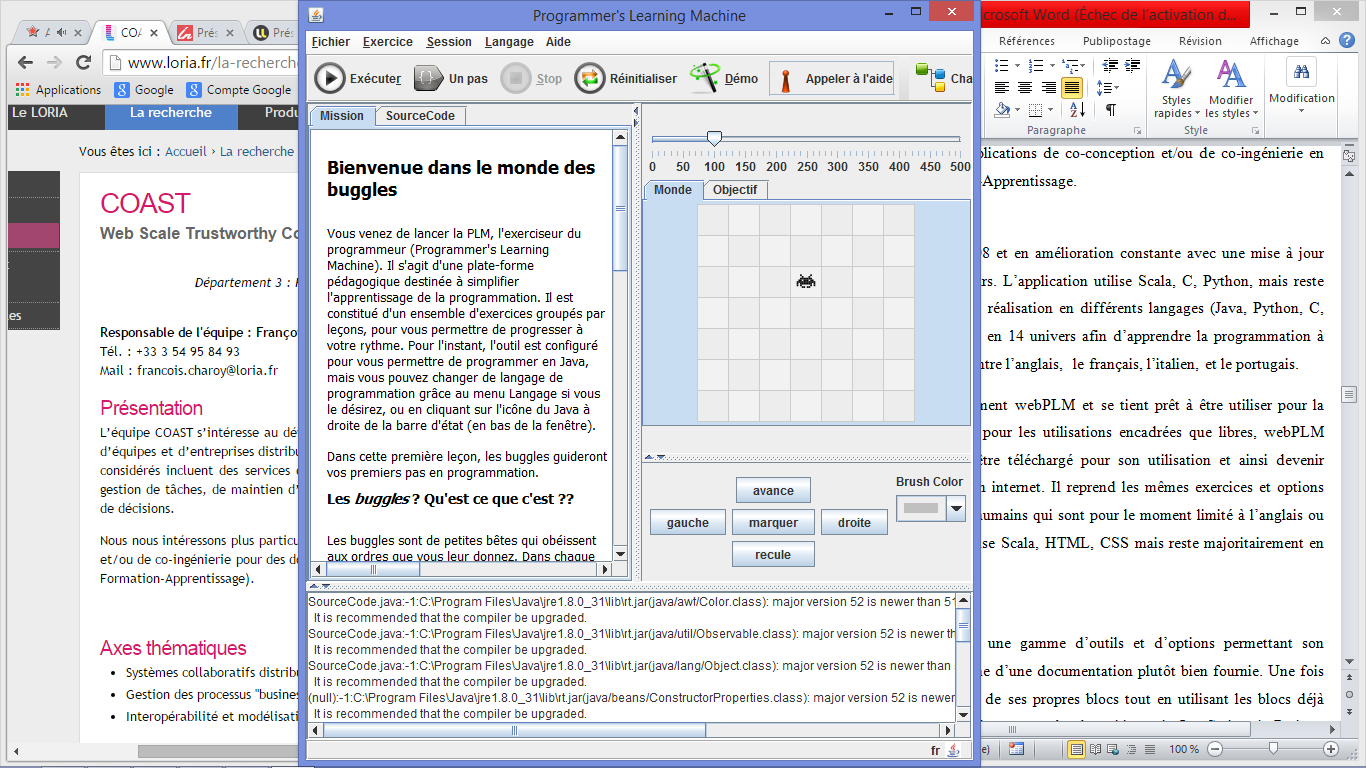


Figure 4 : PLM

Cette année, PLM est petit à petit portée comme service web et se tient prête à être utiliser pour la rentrée 2015. Gain d’accessibilité tant pour les utilisations encadrées que libres. WebPLM souhaite se séparer de la nécessité d’être téléchargée pour son utilisation et ainsi devenir accessible à tout poste disposant d’une connexion internet. Il reprend les mêmes exercices et options que PLM à la différence des langages humains qui sont pour le moment restreints à l’anglais ou au français. L’application web utilise Scala pour le serveur et JavaScript, HTML et CSS.

## Blockly

La librairie Blockly dispose de toute une gamme d’outils, d’options et de documentations permettant son utilisation dans d’autres projets. Une fois en main, il permet une création simple de ses propres blocs tout en utilisant les blocs déjà existant. Blockly peut générer différent langages grâce à ses blocs, du JavaScript, du Python, du Dart et est actuellement en développement pour du PhP. Il dispose également de divers outils notamment sur les traductions et le stockage sur lesquels je me suis basé pour correspondre au besoin et à l’implémentation actuelle de webPLM. Blockly possède également un grand avantage dans l’apprentissage de la programmation, le fait que les erreurs de syntaxe soient impossibles, évitant ainsi de ne pas trouver le point-virgule manquant.

## Outils, technologies et méthodes



Figure 6 : GitHub

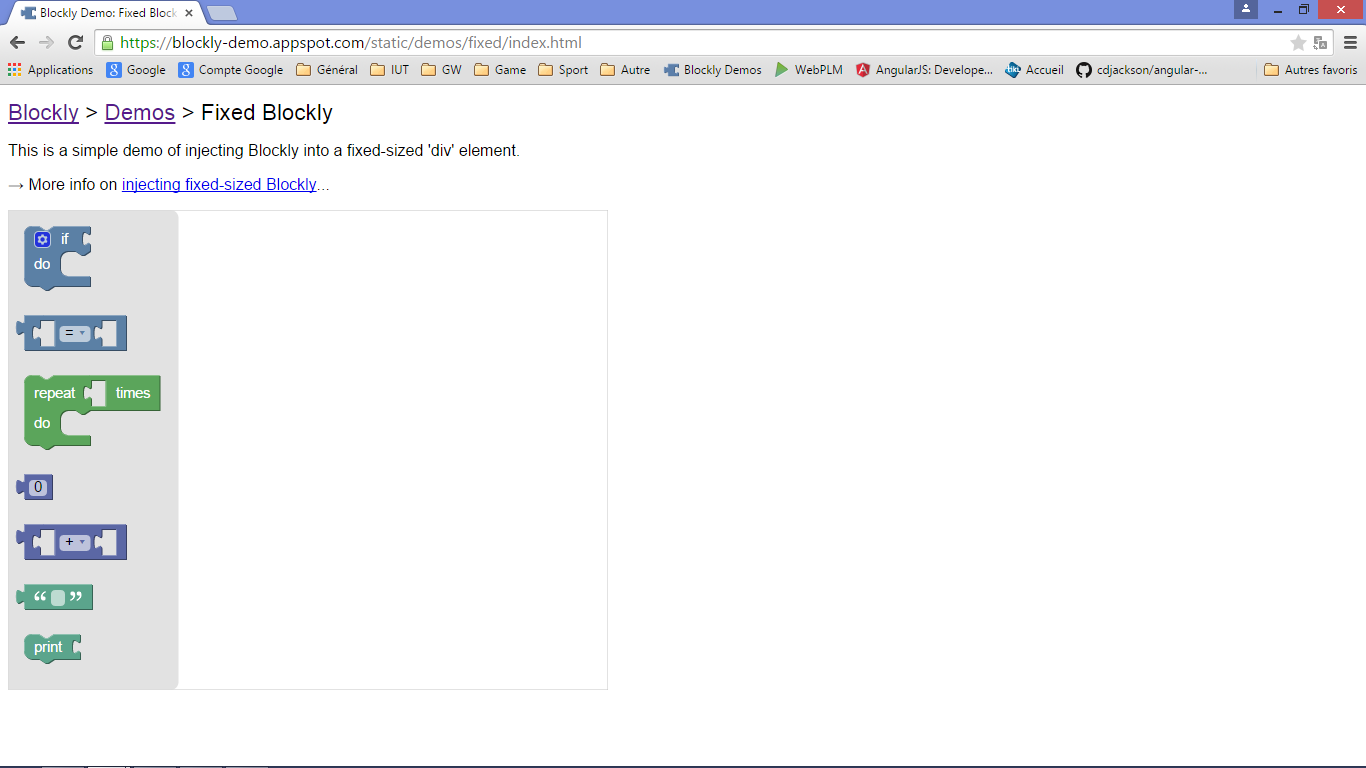


Figure 5 : Blockly

Dans le cadre de mon stage je me suis servi du système de développement collaboratif Git via GitHub où étaient déjà présent les projets PLM, webPLM et Blockly. Et des éditeurs de code Brackets et Eclipse suivant les langages.

Afin de ne pas interférer entre les différentes réalisations développées en parallèles, nous avons tous effectué un doublon du dépôt GitHub principal. Cela nous a permis en effet de développer sans risquer de compromettre le travail du reste de l’équipe et inversement. L’envoi de ma contribution pour son ajout dans le dépôt principal pour la mise en commun s’est réalisé pour ma part vers la fin du stage afin d’avoir une fonctionnalité complète et stable.

Le projet est composé de plusieurs langages de programmation. Dans le cadre de mon contribution j’ai utilisé en très grosse majorité le Java et le JavaScript, les langages principaux de PLM et webPLM.

Pour une organisation et une continuité plus simple dans mon travail il m’a été demandé de réaliser un « reporting », permettant également un suivi plus simple. Il consiste à noter en fin de journée tout ce que j’ai réalisé, les problèmes que j’ai rencontré, et l’évolution de mes objectifs, le tout étant envoyé sur un dépôt GitHub.

# Développement

## Découverte

Dans un premier temps, il a fallu que je découvre l’environnement de travail, PLM, webPLM et Blockly. J’ai donc réalisé quelques exercices sur PLM puis sur webPLM et j’ai terminé sur des utilisations de Blockly via ses démos avant d’explorer le code et sa structure. Le projet utilisant le FrameWork AngularJS, nous avons également réalisé une remise à niveau à travers un tutoriel sur le site Code School.

## Insertion

### Sélection

Pour ajouter Blockly dans l’application il a d’abord fallu l’insérer au même niveau que les autres langages. Comme on peut le voir sur la figure 7 ci-contre, l’éditeur de code sur la partie gauche contient une instruction « forward() ; ». On voit également sur la droite, que nous sommes en Java et qu’un clic sur le langage affiche un panneau qui nous offre le choix entre Java, C, Scala et Python.

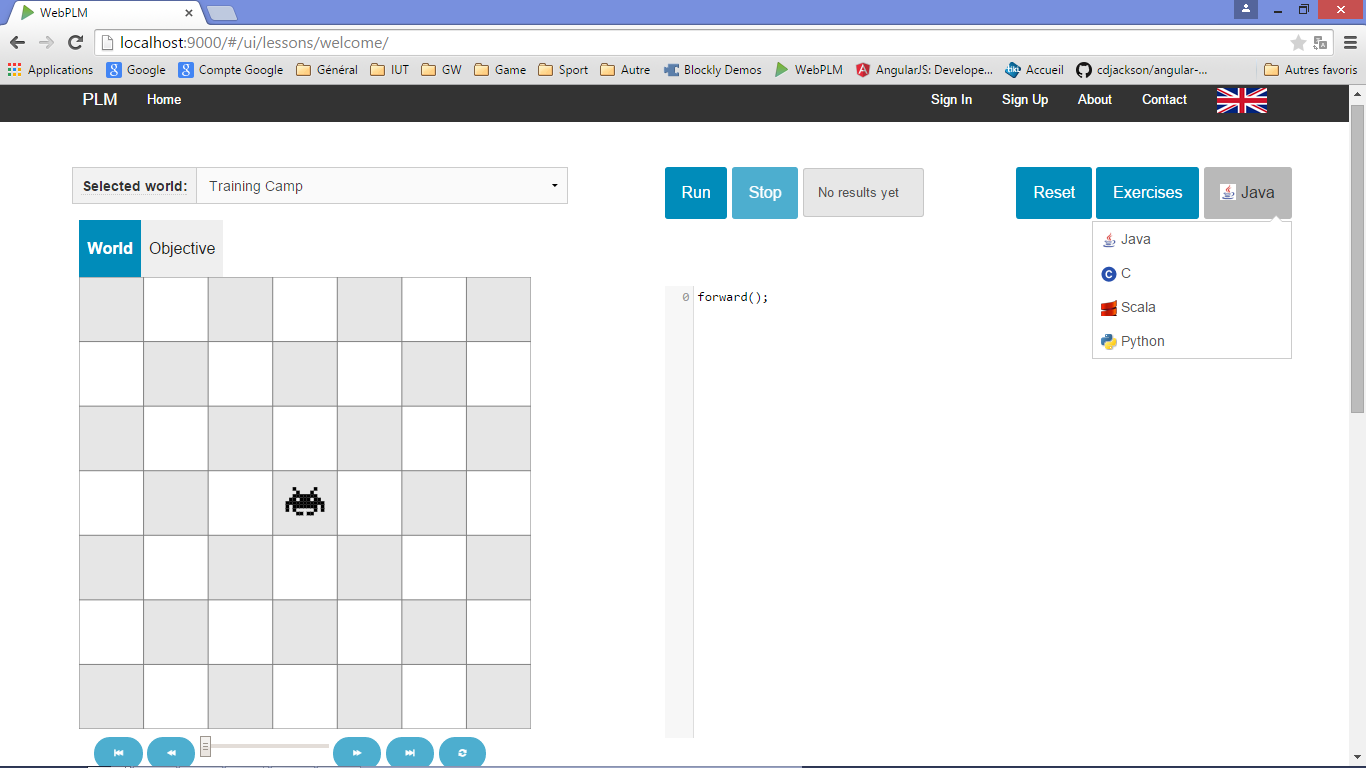


Figure 7 : Affichage de la zone de code et de la sélection du langage avant l’ajout de Blockly

Etant donné que le Python est un langage de résolution d’exercices déjà maîtrisé côté serveur j’ai fait le choix de générer le code de Blockly dans celui-ci, je me suis donc basé sur son implémentation actuelle.

Les choix de langage de programmation disponibles sont chargés en fonctions de différents paramètres :

* Existence d’un fichier nommé Lang*NomLangage*.java qui définit les caractéristiques du langage en question, sur la façon de l’exécuter notamment.
* Existence d’un fichier *NomExercice*Entity.*extensionLangage* dans le package qui définit un exercice dans PLM comme MoriaEntity.java ou encore MoriaEntity.blockly qui indique que l’exercice Moria est réalisable en Java et avec Blockly.

J’ai également dû modifier certains des fichiers principaux qui s’occupent de la gestion des langages comme le fichier « Game.java ». En effet, il possède une liste de tous les langages possibles dans laquelle j’ai dû ajouter le choix de Blockly ou encore l’ajout d’une icône désignant Blockly pour rester dans le même rendu que pour les autres langages. La figure 8 montre le rendu dans le menu déroulant après l’ajout de Blockly, avec les mêmes caractéristiques que les autres langages, le nom et le logo.

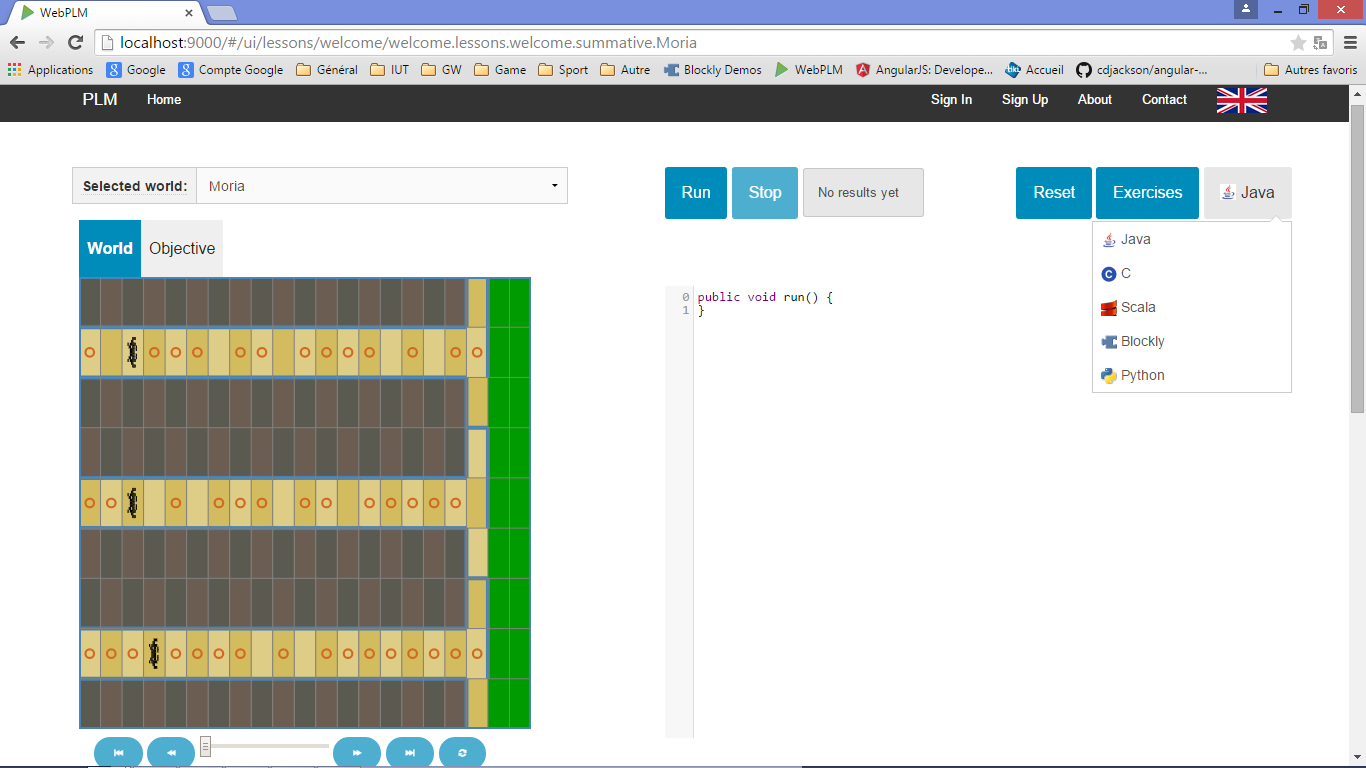


Figure 8 : Sélection du langage après ajout

### Affichage

A ce stade la sélection de Blockly comme langage est devenu possible bien que n’ayant pas de véritable impact sans l’éditeur associé. Il me faut à présent insérer la vue Blockly à la place de la zone de code. Pour ce faire Blockly propose des outils d’insertion, en taille fixe, dynamique ou via une iframe. Ces trois choix, bien que fonctionnelle dans une structure lambda, ne correspondent pas à l’implémentation nécessaire à webPLM, structuré avec AngularJS. Je me suis donc intéressé, après conseil, à une implémentation déjà existante de Blockly via AngularJS partagé sur GitHub.

PLM se sert actuellement de CodeMirror comme interface de codage pour permettre la création d’un code propre pour l’utilisateur c’est-à-dire un code avec une indentation et coloration s’adaptant au langage désiré. J’ai donc dû mettre à l’œuvre mes récentes connaissances en AngularJS afin de modifier la directive affichant CodeMirror et de faire de la place pour créer une directive pour Blockly en m’inspirant du projet AngularJS-Blockly. Pour ce faire je me suis servi, comme vous pouvez le voir sur la figure 9, des options « ng-show » et la variable « exercice.ide » du contrôleur d’exercice désignant si je me sers de CodeMirror ou de Blockly pour éditer le code. Si la condition est validée alors la directive s’affiche.

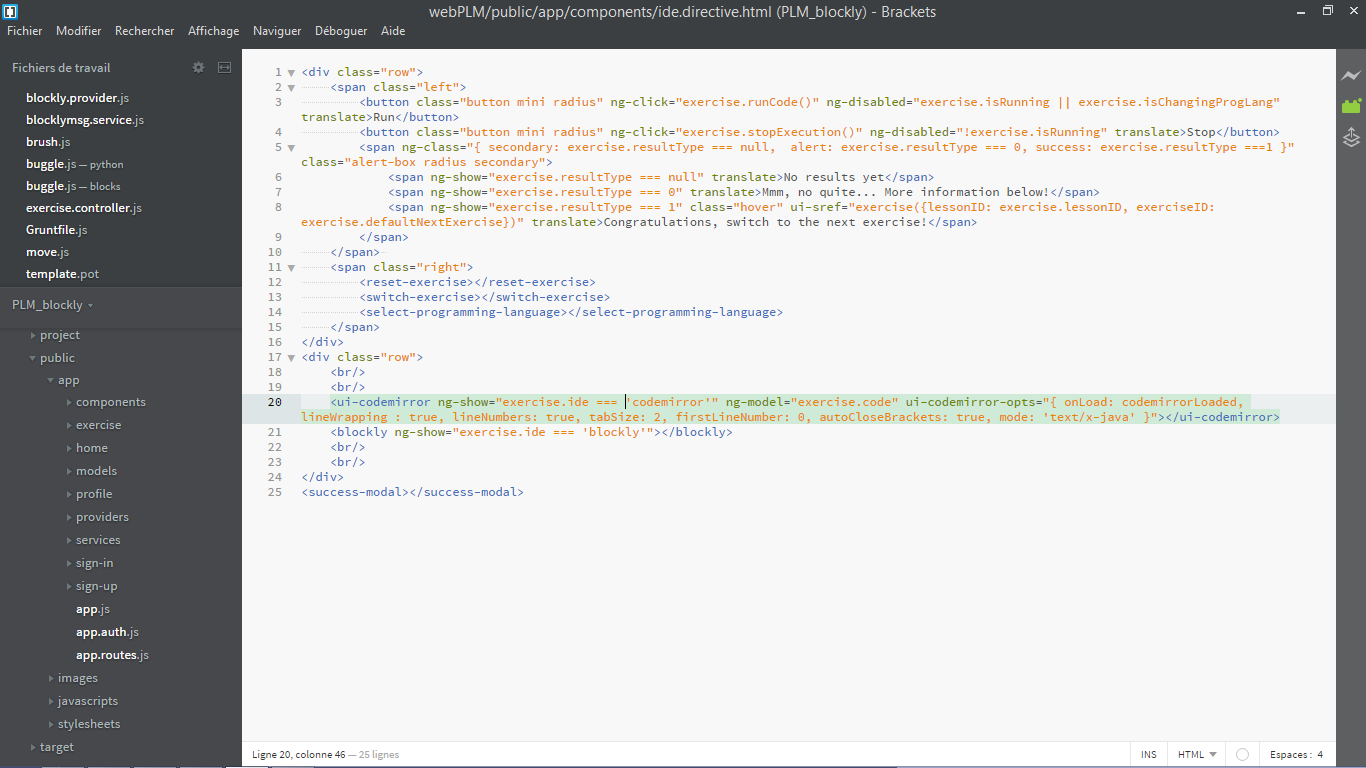


Figure 10 : Partie structure webPLM

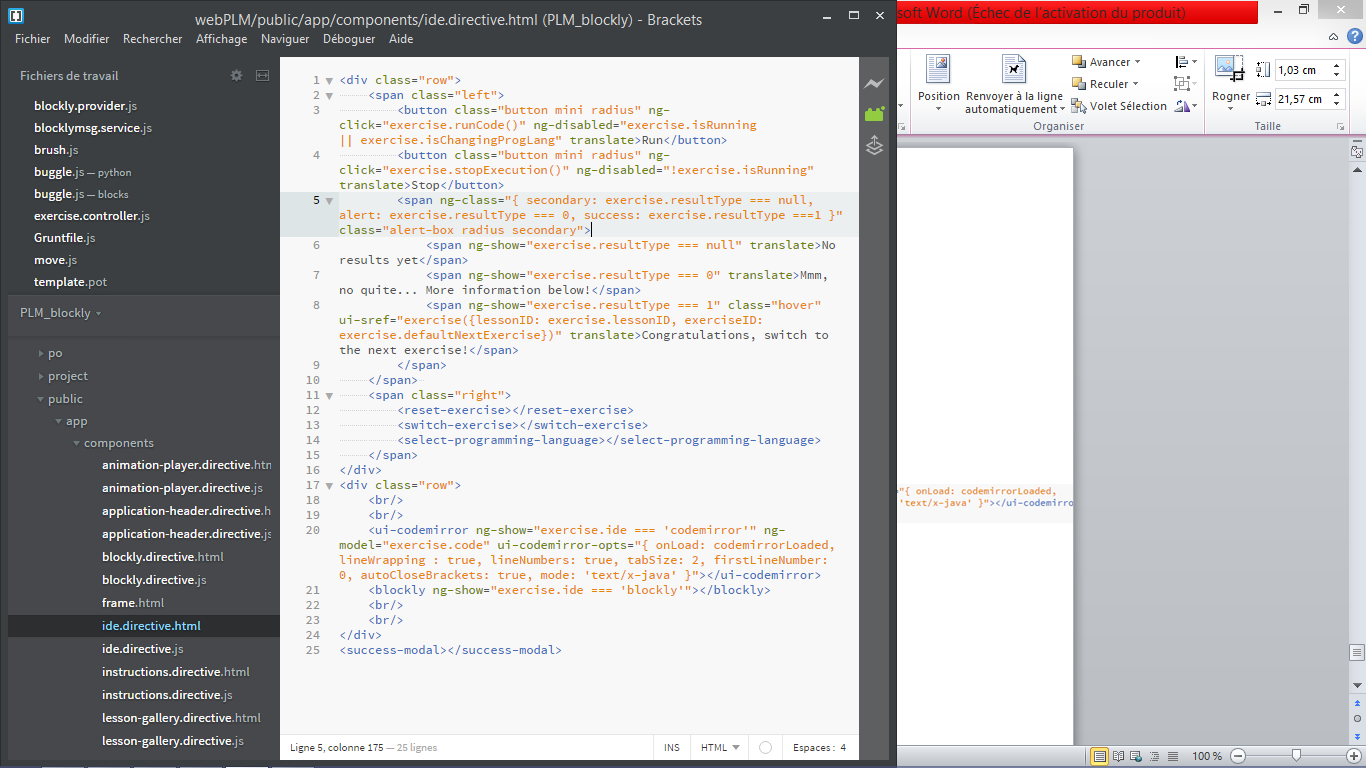


Figure 9 : Partie traitant le choix de l’IDE dans le fichier ide.directive

Puis je me suis servis de l’implémentation AngularJS de Blockly, je l’ai décomposé et restructuré sous forme d’une directive (réf : Annexe 1 – Directive de Blockly) et d’un service (réf : Annexe 2 – Service de Blockly). Afin de distribuer les fonctionnalités à leur place par exemple les services ont été placés dans un fichier blockly.service.js rangé avec les autres services.

C’est à ce moment que j’ai eu des problèmes d’affichage de Blockly. Cela a commencé par l’apparition de Blockly seulement après une modification de la taille du navigateur puis par une augmentation de la hauteur de la zone de travail à chaque itération, clic ou ajout d’un bloc, avec cette dernière ou de changement de taille du navigateur. Pour les corriger, j’ai dû ajouter une contrainte de hauteur fixe de 500px, taille raisonnable étant donné que Blockly intègre deux scrollbars.

### Génération

Cette troisième phase de l’ajout de Blockly consiste à lier la génération de code avec la gestion actuelle de webPLM. C’est dans le fichier exercise.controller.js qu’on définit le fonctionnement principal du client et de sa communication avec le serveur.

Tout d’abord je dois regarder dans la fonction appelée lors de l’exécution, le bouton « Run », (réf : Annexe 2 – Fonction runCode) quel est l’éditeur de code actif entre CodeMirror et Blockly. Dans le cas de CodeMirror rien ne change, je garde l’implémentation actuelle mais dans le cas de Blockly je dois lui demander de générer en Python le code correspondant aux blocs présent dans l’espace de travail. Il possède une fonction « Blockly.Python.workspaceToCode() » qui renvoie le code généré par l’ensemble des blocs de l’espace de travail. Il ne me reste plus qu’à stocker le retour dans la variable recevant habituellement le code extrait de CodeMirror, le reste étant déjà implémenté, le code est envoyé au serveur qui le traite et renvois l’évolution de l’exercice ou les erreurs s’il y en a.

## Toolbox

### Bloc

Dans un deuxième temps, je me suis attelé à la création d’une barre d’outils composée des blocs pour la réalisation de l’exercice en cours. Pour ce faire, j’ai conçu de nouveaux blocs correspondant aux fonctions reconnus par le serveur et nécessaires à Blockly comme les blocs de mouvement « forward », « backward », ou d’interaction avec l’environnement « pickupBaggle », « isFacingWall ».

La documentation Blockly met à disposition une application de Blockly permettant la création de bloc personnalisé, très utile mais assez long. L’interface ci-dessus montre la création du bloc « forward », dans la partie de gauche (**1**) on crée le bloc à l’aide d’une interface Blockly, divers choix sont possible, comme la couleur, le nom, types et nombre des paramètres mais également la valeur de retour ou les connections avec l’instruction précédente ou suivante. La partie de droite se décompose en trois zones. Tout d’abord nous avons l’aperçu (**2**) du bloc visuellement. Puis le code qui crée le bloc (**3**). Et enfin le squelette qui abritera la définition de la génération du code (**4**).

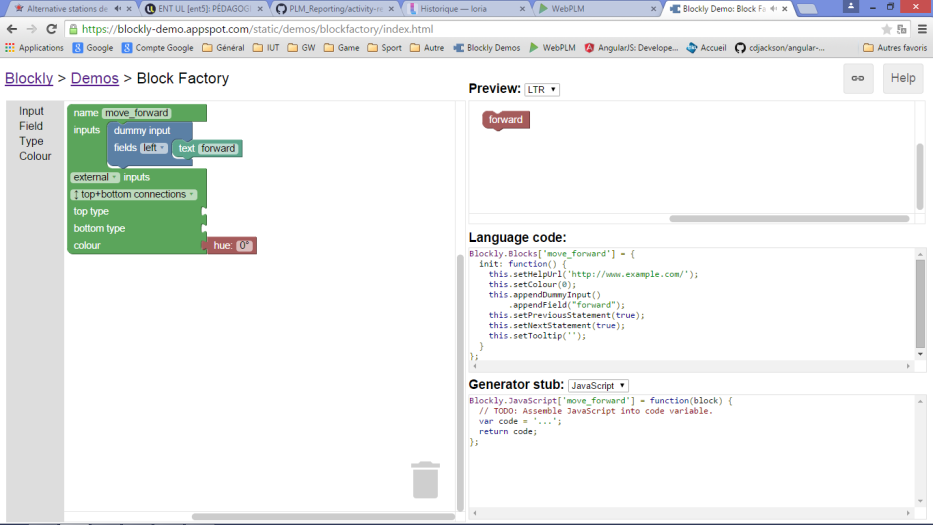


Figure 11 : Aperçu de la création de bloc

**4**

**2**

**3**

**1**

Je l’ai utilisé pour faire le premier bloc et comprendre le fonctionnement d’un bloc afin de réaliser les suivant directement dans le code. Le fonctionnement est le suivant (réf : Annexe 3 – Code de définition du bloc) :

* « This.setColour(0); » Colorie le bloc de la couleur désirée
* « this.appendDummyInput() » Ajout une entrée du type « Dummy » servant uniquement à placer du texte sur le bloc. D’autres entrées pour des valeurs sont possibles
* « .appendField(chaine); » Ajoute un champ de texte.
* « this.setPreviousStatement(true); » Permet une liaison avec un bloc avant.
* « this.setNextStatement(true); » Permet une liaison avec un bloc après.
* « this.setTooltip(chaine); » Défini un texte qui sera afficher si on laisse le curseur sur le bloc pour avoir des informations complémentaires.

La seconde partie de la création d’un bloc est dans le code qu’il doit générer dans chaque langage. Pour notre cas je me suis limité à la génération en Python étant donné que les autres langages ne nous sont pas utiles. Le principe est relativement simple, on prend le squelette donné par la documentation et on ajoute dans la variable « code » ce qu’on souhaite. Pour le bloc « forward » il suffit donc de mettre dans la variable le code « forward()\n » permettant ainsi d’avoir l’appel de fonction reconnu par le serveur et un retour chariot pour l’instruction suivante (réf : Annexe 3 – Code de génération de code en Python).

### Liaison

Les blocs créés il me faut à présent lier la boite d’outils à l’exercice en cours. Pour ce faire je dois créer des fichiers dans PLM pour chaque exercices, et contenant la boite d’outils associée. Je les ai placés au même niveau que les fichiers qui définissent les exercices et ils stockent les boites d’outils au format JSON. Nous avons fait le choix de ne pas rendre ce fichier obligatoire, en cas d’oubli ou de suppression involontaire, donc j’ai placé dans le service de Blockly une boite d’outils par défaut possédant tous les blocs utiles à l'univers Welcome.

Dans notre cas, un fichier JSON définit deux éléments de la boite d'outils dont un seul est indispensable tandis que l'autre permet un meilleur rendu. Tout d'abord il donne le type du bloc (qui peut être comparé à son identifiant), « move\_forward » dans le cas du bloc « forward ». Vu que c'est le client qui possède la librairie Blockly dont la définition des blocs, il est capable de construire le bloc et son fonctionnement uniquement avec ce type. C'est l'élément indispensable. Ensuite on peut choisir de trier ces blocs dans des catégories, par exemple j'ai décidé de placer tous les blocs qui définissent un mouvement pour le Buggle dans une section « Move » dans la boite d'outils. Ce tri est très utile dans notre cas, nous avons besoin d’un nombre plutôt conséquent de blocs, les avoir tous dans la boite d’outils les uns après les autres serai gênant niveau place.

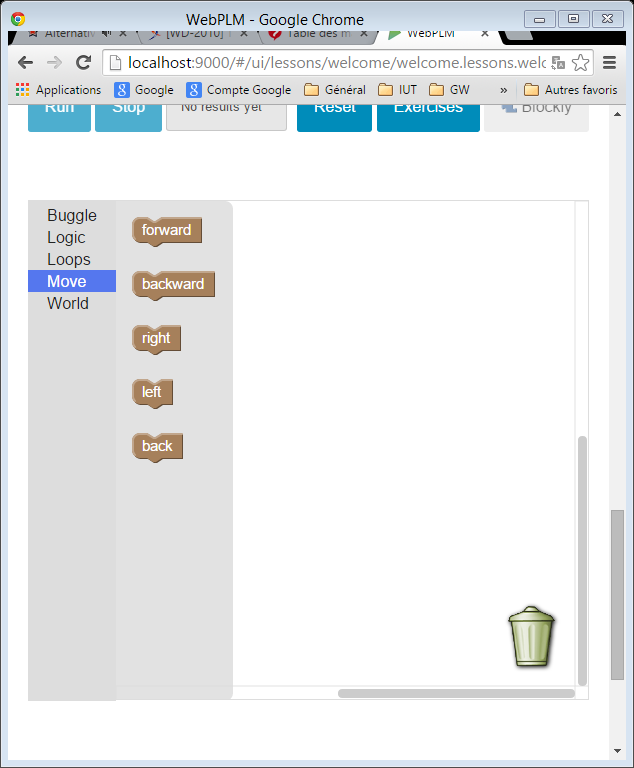
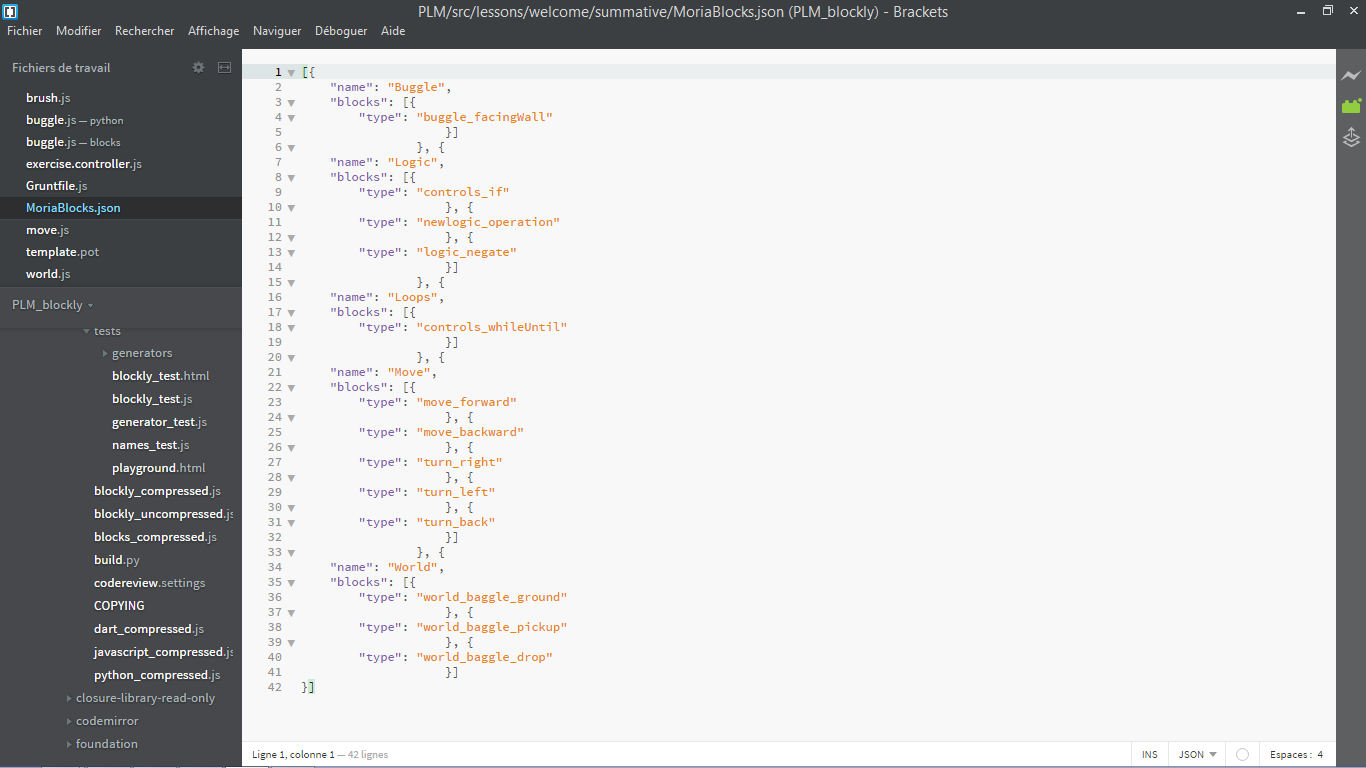


Figure 12 : Blockly avec boite d’outils

Lorsqu'on regarde le fichier de la boite d'outils pour l'exercice Moria on reconnaît d'une part la structure des fichiers JSON mais on peut également voir facilement la structure de la boite d'outils. Les champs « name : » désignent le nom de la section et dans le champ « blocs » se trouve tous les blocs qui iront dans cette section. Cette solution reste relativement simple pour ajouter ou modifier des boites d'outils ou des blocs dans ces dernières.



**Figure 13 : Boite d’outils pour l’exercice Moria**

## Stockage

A ce stade nous pouvons utiliser les blocs, envoyer le code et recevoir la réponse, faire un exercice en somme. Il manque par rapport aux autres langages de programmation le fait de sauvegarder son avancement à chaque exécution pour pouvoir quitter et revenir à l'endroit où on s'était arrêté.

Pour ce faire j'ai dû modifier les envois et traitements entre le client et le serveur. En effet en y réfléchissant on se rend compte que les langages de base disposent d'un unique code qu'il affiche et envois alors que Blockly en compte deux. Le premier est le code que Blockly possède dans son espace de travail et qui est traduit par l'affichage des blocs. Le second en revanche est le code qui est généré à l’exécution. Nous avons donc besoin à présent de manipuler ces deux codes afin de les envoyer au serveur et qu'il soit capable de les traiter tous les deux.

Le projet AngularJS sur Blockly possède un début d'implémentation sur la manipulation de l'espace de travail en format JSON mais n'est malheureusement pas totalement fonctionnel, je me suis donc servis de la manipulation de base disponible dans Blockly en format XML.

La première chose à savoir est le système de sauvegarde. En effet le serveur garde diverses informations sur un total de six fichiers pour Blockly et cinq pour les autres langages, dont tous les noms sont construits suivant la même structure : « *nomUnivers*.lessons.*nomUnivers*.*typeExercice*.*nomExercice*.*extensionLangage*.*typeContenu »*. Je vais pour la suite prendre pour exemple l’exercice Moria réalisé sous Blockly dont les fichiers de sauvegarde possède tous une partie commune, jusqu’au type du contenu, soit « welcome.lessons.welcome.summative.Moria.blockly » traduit dans la suite par « # ».

* « #.code » contient le code de résolution de l’exercice
* « #.workspace » sauvegarde l’espace de travail
* « #.correction » garde la solution pour un exercice réussi
* « #.error » stock les erreurs retournées par le serveur
* « #.done » sait si l’utilisateur a réussi l’exercice par le passé
* « #.mission » affiche l’énoncé de l’exercice

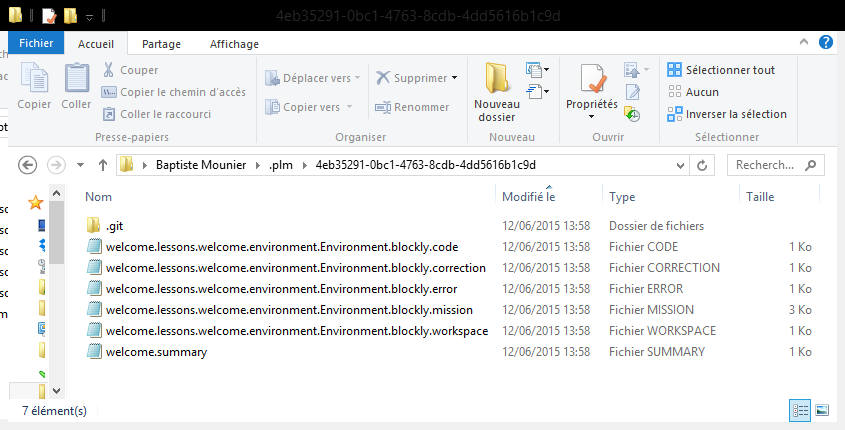


Figure 15 : fichiers d’une session pour un exercice

Le tout étant soit envoyé sur un dépôt git soit, si aucune connexion internet, stocké dans un dossier « .plm » dans l’espace utilisateur de la machine en local.

Pour effectuer cette réalisation, j’ai dû modifier le trajet de l’envoi et de prise en charge. Tout d’abord, il a fallu, dans la fonction runCode (réf : Annexe 2 – Fonction runCode) de exercise.controller.js, extraire le contenu de l’espace de travail en XML avec les fonctions de Blockly.Xml.workspaceToDom et Blockly.Xml.domToText. La première prend le contenu de l’espace de travail et le transforme en un élément du DOM permettant ainsi l’utilisation de la seconde qui crée une chaine de caractère à partir d’un élément du DOM. Il me reste plus qu’à insérer ce texte dans un nouveau champ de l’objet envoyé au serveur.

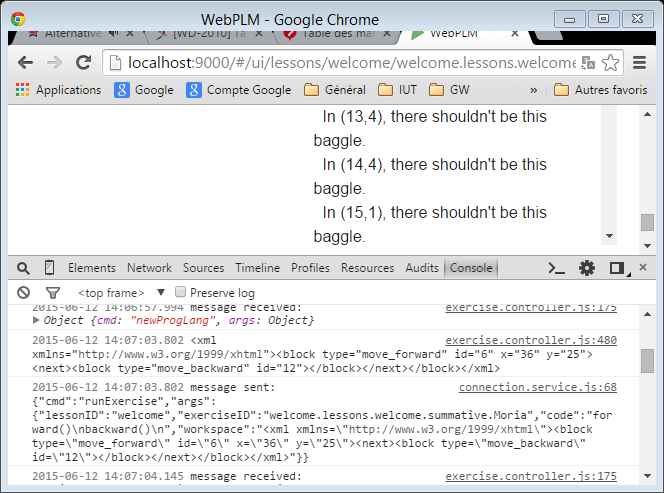


Figure 14 : XML généré pour un bloc forward suivi d’un bloc backward

Une fois reçut, le serveur réparti le contenu de l’objet dans les différents fichiers. La classe PLM en scala s’occupe de cette partie mais elle nécessite la création préalable de ces fichiers qu’elle ne fait que remplir. C’est dans « Exercise.java » que sont générés les fichiers, et j’ai ajoute une condition sous laquelle on crée deux fichiers pour Blockly et un pour les autres langages. De ce fait, seul les exécutions en Blockly provoqueront la création d’un fichier en « #.workspace ».

## Traduction

Le système de traduction actuel est plutôt complexe. Le projet dispose de plusieurs mécanismes de traductions, qui seront mis en commun prochainement. Et Blockly possède le sien également. L’idéal est de centraliser ces trois systèmes ou du moins d’ajouter celui des blocs à l’un ou l’autre existant.

Je me suis penché avant tout sur le fonctionnement de celui de Blockly qui s’est révélé très complet au niveau des langues disponibles, plus de 40 actuellement. En revanche, il pose un problème majeur, il se sert d’un rechargement complet de la page afin de prendre en compte le changement de langue. Système qui ne convient pas du tout à l’implémentation du projet. Son association dans un mécanisme de traduction existant du projet est donc devenue l’unique choix.

WebPLM se sert de gettext pour la traduction. J’ai donc implémenté une solution pour y associer Blockly également. Le principe est simple, remplacer toutes les chaines de caractères par un appel d’une constante contenant le texte de la langue en cours par l’appel « Blockly.Msg.MOVE\_FORWARD\_TITLE ». A chaque changement de langue, toutes les variables sont regénérées avec le contenu dans la bonne traduction grâce à gettext qui regarde le langage et la chaine qu’on lui donne et qui retourne la traduction. PLM se sert de l’anglais comme référence, c’est-à-dire que si aucune traduction n’a été trouvée il renvois l’état de base de la variable qui est en anglais. Pour ma part, je n’ai pas modifié ce comportement déjà fonctionnel. J’ai ajouté un service BlocklyMsg regroupant toutes les variables de traductions utilisées par Blockly et permettant la liaison avec gettext. Et également réalisé une première traduction, dans le dictionnaire, des mots que j’ai ajouté et je me suis servis des traductions de Blockly pour les blocs de base.

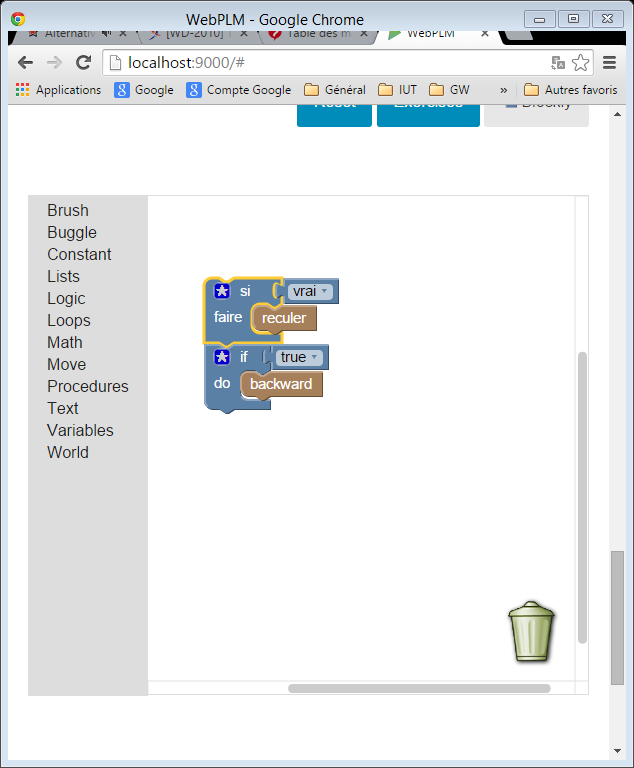


Figure 16 : Blockly avec du français et de l’anglais

En ce qui concerne le changement de langue en cours de réalisation d’un exercice, le fonctionnement est particulier. En effet, toute la page se mets à jours grâce au système de gettext sauf en ce qui concerne Blockly. Lors de la notification du changement du langage, on ne recharge pas la page, mais en revanche on peut actualiser la boite d’outils. L’espace de travail n’est par contre pas remis à jours, il se peut alors que l’utilisateur se retrouve avec des blocs anglais et français, chose pas gênante étant donné que la génération de code pour l’exécution et le stockage est le même quel que soit la langue. Si l’utilisateur change d’avis en cours de réalisation et trouve le mélange gênant, il lui suffit de lancer l’exécution pour sauvegarder ses blocs et de recharger la page dans la langue de son choix afin que tous les blocs s’uniformisent.

# Conclusion

Je suis parti d’un projet en cours de réalisation, le portage d’une application en client lourd à un service web. Mon travail consistait à ajouter une fonctionnalité. L’usage de Blockly, un éditeur visuel de code, parmis les langages disponibles pour l’exerciseur.

Ma contribution c’est déroulé en cinq étapes majeures. Une phase de découverte pour m’adapter à mon environnement de travail : projet et langages. Puis l’ajout de l’accès de la nouvelle fonctionnalité. Suivi de la création de l’environnement de cette dernière. Et de la mise en place du fonctionnement derrière celle-ci. Enfin d’une touche plus cosmétique mais importante.

Je suis donc arrivé à un exerciseur où Blockly est fonctionnel pour la réalisation des exercices du premier univers, mais aussi dont l’ajout pour les univers suivant ne posera probablement pas de problème à l’avenir.

La traduction des noms de sections serait un atout, déjà sur le plan de l’uniformisation de la fonctionnalité. Un effet de coloration du bloc provoquant l’effet de l’aperçu de l’exercice serait une très profitable amélioration mais risque de poser quelques problèmes, notamment vu que l’aperçu ne verrouille pas l’éditeur de code. Une mise en place de boite d’outils par pallier évitant ainsi de surcharger la boite d’outils de bloc des autres univers lorsqu’on oublie de définir la boite pour l’exercice en cours. Et pour finir ou plutôt pour commencer, une batterie de tests serait des plus judicieuses afin d’éviter tout développement régressif.

# Bibliographie

## Sites web

Code School, 7 avril, <https://www.codeschool.com>

LORIA, <http://www.loria.fr/les-actus>

PLM, <http://www.loria.fr/~quinson/Teaching/PLM>

Blockly, <https://developers.google.com/blockly>

GitHub, <https://github.com/BaptisteMounier>

AngularJS, <https://github.com/cdjackson/angular-blockly>

CodeMirror, <https://codemirror.net>

## Rapport

Mounier Loïc, Développeur Web, 25 octobre 2010 au 15 janvier 2011, Bibliothèque Universitaire IUT Nancy-Charlemagne, DUT Informatique, Université de Lorraine, 46 pages

Table des annexes

Annexe 1 :

Diagramme de Gantt

Organigramme 2014 LORIA

Directive de Blockly

Annexe 2 :

Service de Blockly

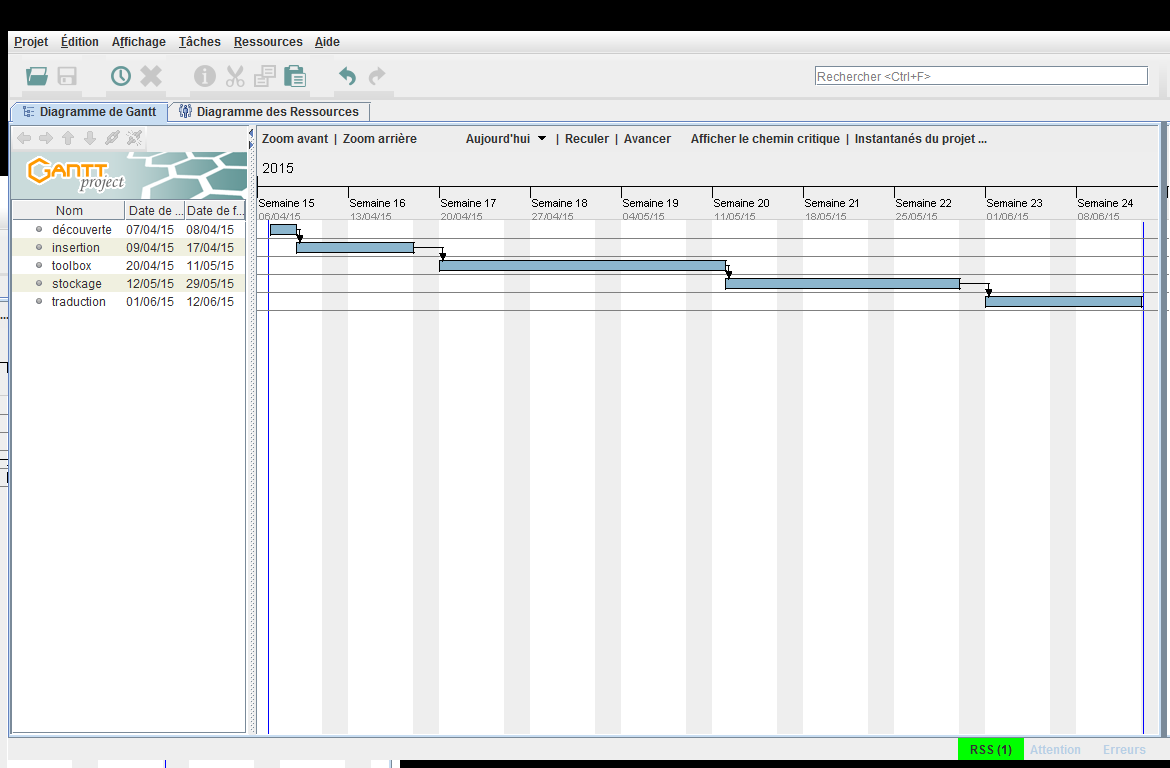
Fonction runCode

Annexe 3 :

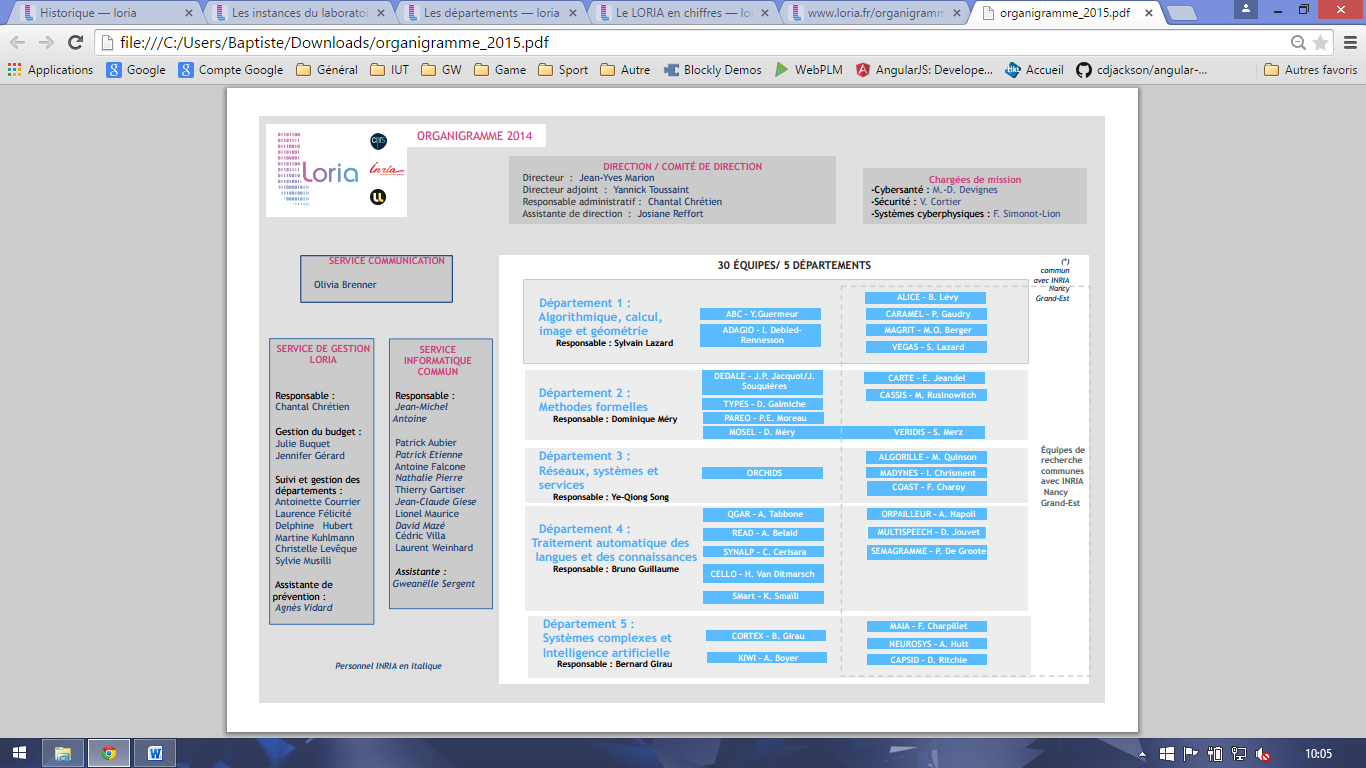
Génération de code en Python

Définition du bloc forward

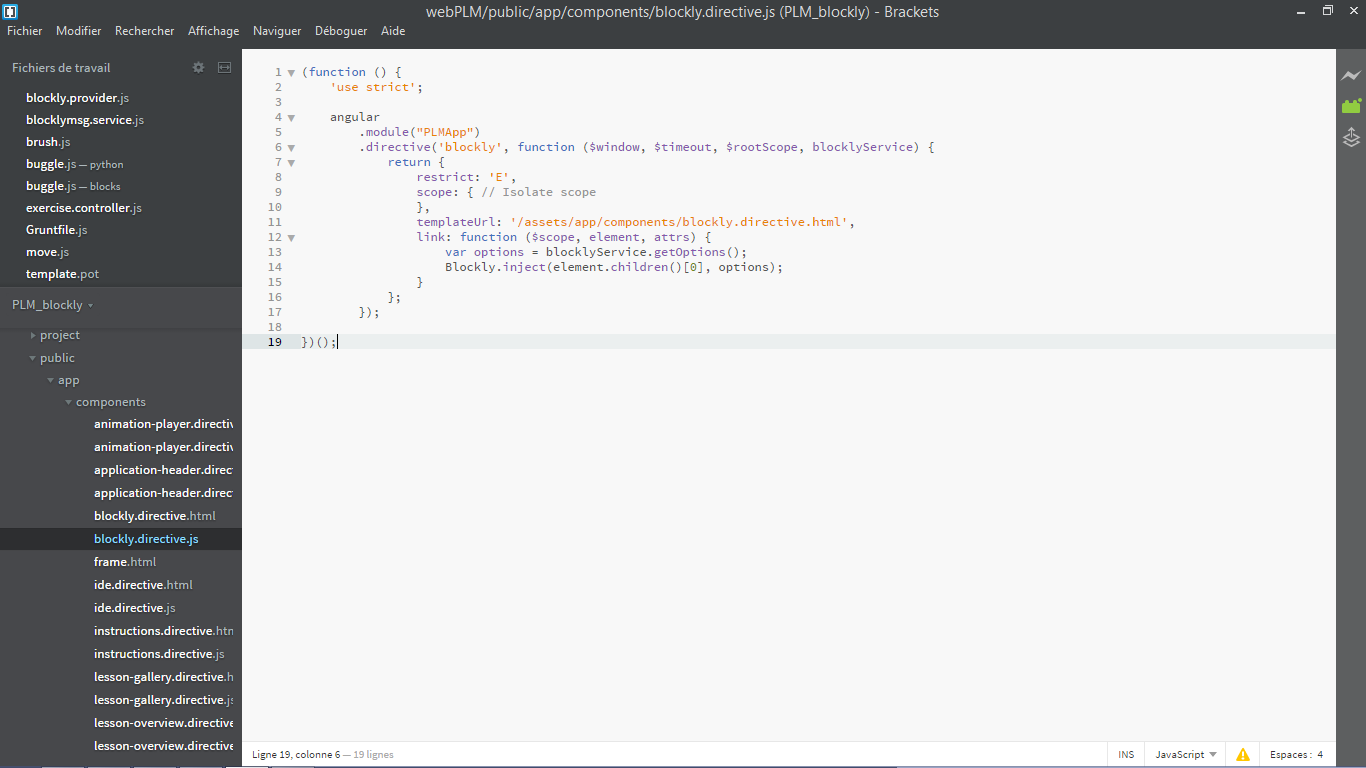
**Annexe 1**

Diagramme de Gantt

Organigramme 2014 LORIA

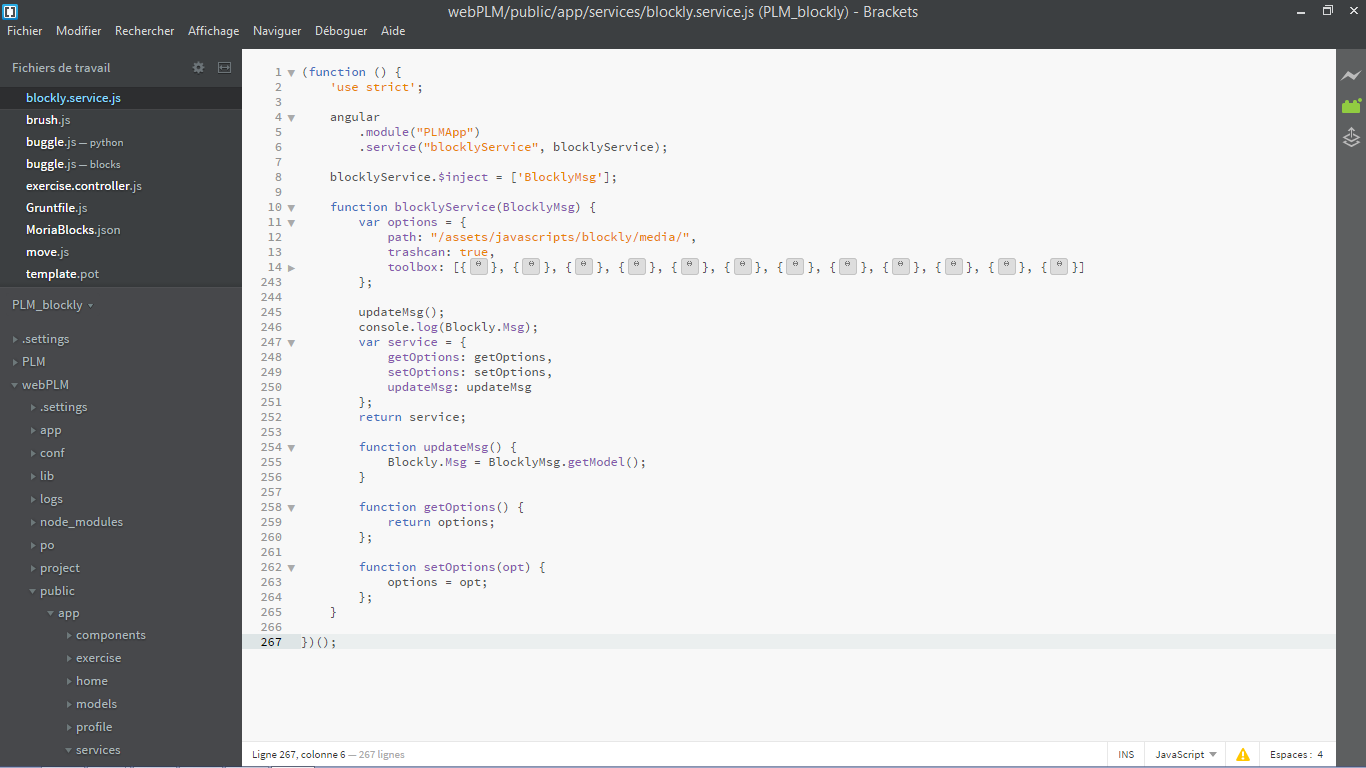


Directive de Blockly



**Annexe 2**

Service de Blockly

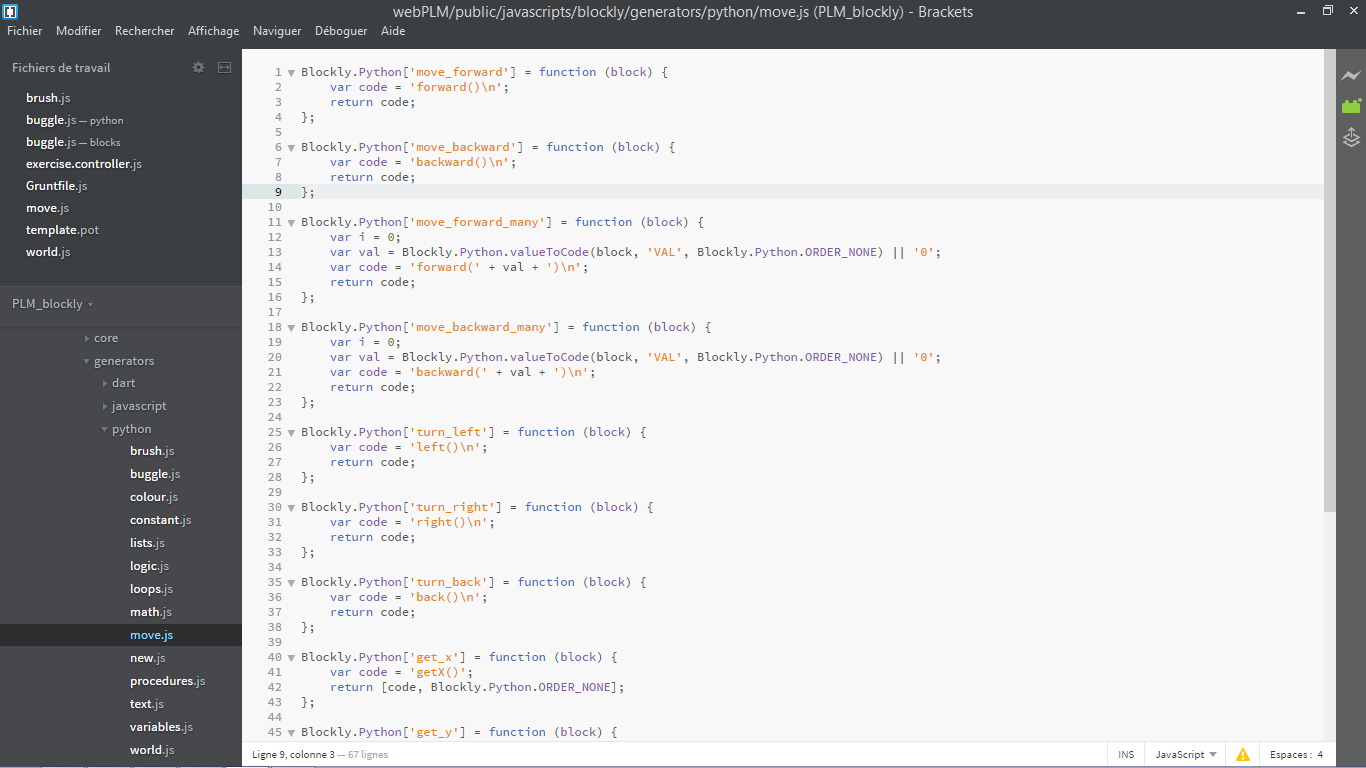


Fonction runCode

****

**Annexe 3**

Génération de code en Python



Définition du block forward

