Mémoire d'ingénieur

Hervé Bühler

Année 2014–2015

Stage de fin d'études réalisé dans l'entreprise LORIA

en vue de l'obtention du diplôme d'ingénieur de TELECOM Nancy

Maître de stage : Martin Quinson

Encadrant universitaire : Gerald Oster

Déclaration sur l'honneur de non-plagiat

Je soussigné(e),

Nom, prénom : Bühler, Hervé

Élève-ingénieur(e) régulièrement inscrit(e) en 3e année à TELECOM Nancy

Numéro de carte de l'étudiant(e) : 2014041022

Année universitaire : 2014–2015

Auteur(e) du document, mémoire, rapport ou code informatique intitulé :

Winter is Coming – You know nothing Jon Snow

Par la présente, je déclare m'être informé(e) sur les différentes formes de plagiat existantes et sur les techniques et normes de citation et référence.

Je déclare en outre que le travail rendu est un travail original, issu de ma réflexion personnelle, et qu'il a été rédigé entièrement par mes soins. J'affirme n'avoir ni contrefait, ni falsifié, ni copié tout ou partie de l'œuvre d'autrui, en particulier texte ou code informatique, dans le but de me l'accaparer.

Je certifie donc que toutes formulations, idées, recherches, raisonnements, analyses, programmes, schémas ou autre créations, figurant dans le document et empruntés à un tiers, sont clairement signalés comme tels, selon les usages en vigueur.

Je suis conscient(e) que le fait de ne pas citer une source ou de ne pas la citer clairement et complètement est constitutif de plagiat, que le plagiat est considéré comme une faute grave au sein de l'Université, et qu'en cas de manquement aux règles en la matière, j'encourrais des poursuites non seulement devant la commission de discipline de l'établissement mais également devant les tribunaux de la République Française.

Fait à Paris, le 13 novembre 2015

Signature :

Mémoire d'ingénieur

Hervé Bühler

Année 2014–2015

Stage de fin d'études réalisé dans l'entreprise LORIA

en vue de l'obtention du diplôme d'ingénieur de TELECOM Nancy

Hervé Bühler
16, allée de fontainebleau
75019, PARIS
06 23 64 95 77

TELECOM Nancy
193 avenue Paul Muller,
CS 90172, VILLERS-LÈS-NANCY
+33 (0)3 83 68 26 00
contact@telecomnancy.eu

LORIA
Campus scientifique BP 239
54506 Vandoeuvre-lès-Nancy
+33 3 83 59 20 00

Maître de stage : Martin Quinson

Encadrant universitaire : Gerald Oster

Remerciements

"Night gathers, and now my watch begins.
It shall not end until my death.

I shall take no wife, hold no lands, father no children.
I shall wear no crowns and win no glory.
I shall live and die at my post.

I am the sword in the darkness.
I am the watcher on the walls.
I am the shield that guards the realms of men.

I pledge my life and honor to the Night's Watch,
for this night and all the nights to come."

– The Night's Watch oath

Table des matières

1 Introduction

Internet est un outil efficace pour diffuser savoirs et connaissances. Nous avons ainsi accès à des encyclopédies, des tutoriels, des cours et explications en tout genre, fournit par des amateurs ou des professionnels. Ce potentiel est entre autre exploité dans des cours en ligne : en particulier depuis quelques années, les MOOCs sont apparus et se sont répandus. Un MOOC, pour Massive Open Online Course, peut être à propos d'à peu près n'importe quel sujet ; et la forme peut également varier. Certains sont proches de simples cours statiques, d'autres auront un ou plusieurs professeurs attitrés. Certains MOOC chercheront à valider et certifié les compétences acquises par les étudiants, de manière assez proche d'un enseignement classique D'autres laissent le choix aux étudiants de l'objectif d'apprentissage.

Les MOOCs présentent de nombreux attraits. Principalement, ils sont très accessible : ils ne nécessitent qu'un accès internet, et n'ont pas ou peu de contrainte sur les horaires. C'est particulièrement intéressant pour un public actifs et non étudiant. Par ailleurs, un MOOC permet à un professeur d'avoir un grand nombre d'élève.

Si chaque cours pourra être organisé différemment, cette dernière caractéristique est cependant contraignante : le nombre d'élève, disproportionné par rapport au nombre d'enseignant, empêche un suivi traditionnel. On peut atteindre plus de 100 000 élèves dans le cas de certains cours anglophone. Il devient très compliqué, voir impossible de fournir une aide adapté aux étudiants le cas échéant.

Une solution peut alors être de chercher des outils informatique pouvant aider ou dans certain cas remplacer l'enseignant. Le format informatique du cours permet par ailleurs une intégration plus aisée d'un tel outil dans le cours.

Il faut alors déterminer un objectif pertinent pour un tel outil. On peut essayer de sélectionner des exercices intéressant pour l'élève, d'évaluer son niveau, de lui fournir des conseils automatique,.... Les possibilités sont nombreuses, mais il faut concilier un point de vue pédagogique, ce qui sera utile pour l'élève et un point de vue technique, ce qui sera réalisable au vu des informations que l'on pourra exploiter.

Nous nous intéresserons ici au cours informatique, en particulier ceux où l'élève a à écrire un programme pour répondre à un exercice. C'est en effet un contexte propice pour développer un tel outil. Nous avons ainsi accès à un grand nombre d'information qu'un ordinateur pourra analyser plutôt aisément. On peut automatiquement évaluer un exercice (est-ce que le programme écrit compile, fait ce qui est demandé ?), et le progrès de l'élève est visible au fur et à mesure de son travail. Ce progrès peut se voir sous la forme de débuggage technique (Des erreurs de syntaxe corrigées au fur et à mesure) ou de progrès fonctionnel (Gérer une partie du problème avant de généraliser,...). Le code ne reflète pas nécessairement toute la réflexion de l'élève, mais reste généralement informatif. En théorie, les méthodes applicables à ce contexte sont transposable, mais il faudrait développer un moyen d'évaluer de la même manière des exercices de types différents,

dans d'autres matières.

En général, il n'est pas possible d'évaluer une réponse "libre", et un QCM, par exemple, ne présentera pas assez d'information. Par ailleurs, il est rare que la soumission d'un élève reflète l'évolution de son travail : la réflexion se fait souvent en amont, et la réponse finale ne permet pas de déduire les égarements et essai intermédiaire.

La problématique qui a été choisit sera de détecter des élèves en difficulté, afin d'ensuite pouvoir intervenir et leur fournir une aide adaptée. Plusieurs problème surviennent alors.

La définition d'en difficulté, tout d'abord, peut être délicate. If faut d'abord savoir quel est le but du cours pour les étudiants. Dans certains cas, se sera la réutilisation professionnel des connaissances acquises, ce qui sera quasiment impossible à mesurer. Dans d'autres, cela pourra être une utilisation quotidienne (langues,...), une meilleure culture générale,... Idéalement, il faudrait pouvoir suivre des étudiants sur long termes, afin de pouvoir vérifier si tel ou tel signes correspondait effectivement à un comportement à risques d'après les critères choisit. Il nous faut cependant trouver un moyen d'évaluer les élèves plus raisonnable. Assez logiquement, les notes d'examens devraient être un indicateur correct ; de même, on pourra évaluer d'éventuels exercices intermédiaire. Il faut cependant garder à l'esprit que cet indicateur sera choisit, et ce choix sera au moins partiellement arbitraire. Notre outil, au mieux, saura détecter les élèves correspondant à l'indicateur ; il faudrait une étude pédagogique pour confirmer que ce critère est pertinent.

Par ailleurs, le but exact est de savoir quel élève a besoin d'une aide, et quand. Un élève en tête de classe, par exemple, pourrait dans certains cas profiter grandement d'une intervention sur un exercice avancé. L'intérêt de l'intervention n'est pas nécessairement lié au niveau de l'élève. La question est également de savoir comment l'on veut répartir les ressources de l'enseignant si elle sont limité : dans le cas d'un système automatique, dispenser des conseils ne pose pas de problème, mais si le l'enseignant doit intervenir lui-même, aider un élève signifie qu'un autre ne profitera pas de cette attention. Idéalement, il faudrait mesurer l'efficacité d'une intervention. On pourrait par exemple tester un outils ou une méthode sur une partie d'une classe, et laisser les autres élèves sans aide. Cela demanderait des résoudre une série de problème. Ethique, tout d'abord, puisque qu'il faudrait nécessairement pénaliser une partie des élèves. Mais également des problèmes de biais (Un enseignant croyant dans une méthode risque de mieux l'appliquer) ou tout simplement, de moyen. Il faudrait en effet avoir un grand nombre d'élève, volontaire,....

La seule solution en l'occurrence pour définir "en difficulté" est au final d'imposer une définition. Le but sera d'avoir des éléments mesurable, afin de pouvoir exploiter cette définition, tout en étant crédible d'un point de vue pédagogique.

2 Présentation de l'entreprise

Le LORIA, laboratoire lorrain de recherche en informatique et ses applications, a pour mission la recherche fondamentale et appliquée en sciences informatiques. Il a été fondé en 1997. C'est une Unité Mixte de Recherche commune à plusieurs établissements : le CNRS, l'Université de Lorraine et l'Inria.

Le LORIA est l'un des plus grand laboratoire de Lorraine, avec plus de 450 personnes répartie dans 30 équipes structurés selon cinq départements :

- Algorithmique, calcul, image et géométrie
- Méthodes formelles
- Réseaux, systèmes et services
- Traitement automatique des langues et des connaissances
- Systèmes complexes et intelligence artificielle

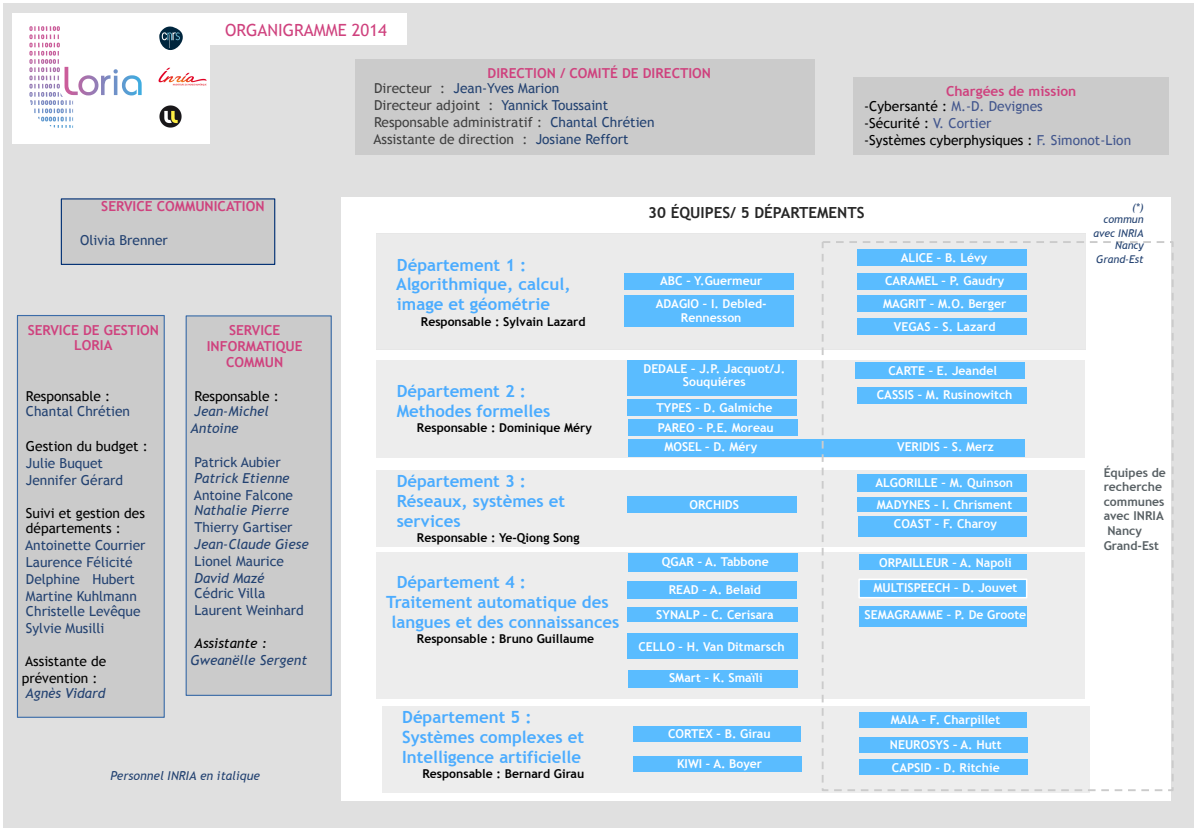

FIGURE 2.1 – Organigramme du Loria, 2015

Le projet PLM qui nous intéressera est cependant un peu part : mené par Géral Oster et Martin Quinson, il se trouve entre deux équipes et entre deux département. Gérald Oster fait partie de l'équipe Coast, du département Réseaux, systèmes et services, et Martin Quinson de l'équipe Véridis, du département Méthodes formelles.

Véridis est une équipe experte en méthodes formelles et en vérification, travaillant principalement dans le domaine des algorithmes et systèmes parallèles et répartis. Coast s'intéresse au développement de services pour l'hébergement d'équipes et d'entreprises distribuées (ou virtuelles) sur Internet.

Le projet PLM n'est donc pas particulièrement lié à ces équipes. En plus de Gérald Oster et Martin Quinson, l'équipe du projet est complété par Matthieu Nicolas, en tant qu'ingénieur, et ponctuellement aidé par des stagiaires.

3 Contexte

3.1 PLM

La PLM, pour Programmer Learning Machine, est un logiciel d'apprentissage de la programmation. Il est principalement utilisé par les élèves de première année de TELECOM Nancy, et a été développé pour ce public. Le projet a été initié par Gérald Oster et Martin Quinson en 2008, et a bien évolué depuis. Le dernier changement majeur fut le passage d'une structure de client lourd vers une architecture client-serveur web. Cette dernière version est celle qui a été utilisé pour la rentrée 2015, et est le fruit du travail de Matthieu Nicolas.

La PLM est multi-plateforme, gratuite, et propose de résoudre les exercices dans différents langages : Java, Python, Scala, Blockly. Le C fut un temps utilisable dans une version expérimentale, mais n'a pas encore été ré-intégré dans la version actuelle.

La PLM se présente comme un environnement de développement simplifié, dans lequel l'élève peut résoudre les exercices qui lui sont proposé. Différentes leçons sont proposées, de la plus basique, welcome, qui reprend les bases de la programmation, à certains concept plus avancés, comme les tris. Chaque exercice est associé à son énoncé, son API (la liste des fonctions utilisables dans le "monde" correspondant), et son monde, donc. Par exemple, dans beaucoup d'exercice, il sera question de déplacer un pion (un buggle pour être exact) sur un quadrillage, et de lui faire effectuer différente tâche. Cet aspect visuel permet d'avoir un retour immédiat sur l'action du code écrit, facilitant ainsi l'apprentissage. L'aspect ludique n'est pas non plus à négliger.

La PLM est également prévu pour fournir des traces de l'activité des élèves : ces traces, locales, peuvent servir pour évaluer le travail effectué. Elles peuvent également être publiées publiquement si l'élève y consent. Elles sont alors anonymisées. C'est ces traces que nous allons utiliser pour analyser le comportement des élèves et essayer de détecter des difficultés.

La PLM est un projet libre (sous license GPL GNU pour être précis), et parfaitement utilisable au dehors de TELECOM Nancy. De ce fait, si la majorité des utilisateurs et des traces associées sont des étudiants de première année, nous disposons également de traces d'utilisateurs autres, sans plus d'informations sur ceux-ci.

3.2 TELECOM Nancy

TELECOM Nancy est une école qui forme des ingénieurs généralistes en informatique et technologie du numérique. TELECOM Nancy fait partie de l'université de Lorraine, et est associé, comme son nom l'indique, à l'institut Mines-Télécom. Elle forme en trois ans des élèves recruté à bac + 2. Ces élèves viennent principalement d'IUT d'informatique ou des classes préparatoires.

La PLM est utilisé en première année, dans deux modules. Le premier consiste en une formation à l'informatique des élèves issus de classes préparatoire. Il est à noter que jusqu'à l'année précédente, l'informatique en classe préparatoire était une option, plutôt orienté sur l'algorithmique. La nouvelle et les futures générations d'élèves, cependant, auront tous eut une formation informatique à leur entrée à l'école. La PLM est également utilisé par la suite dans le module de TOP, techniques et outils pour programmer.

Ce qui nous intéressera particulièrement ici est le module d'initiation. Ce module dure 30 heures répartie sur deux semaines, et a pour but de fournir les bases du langages Scala aux élèves. Une longue série d'exercice y est proposé, proposant un apprentissage de la PLM elle-même, des bases de l'informatique, mais également des exercices plus avancés. A la fin des deux semaines, les étudiants doivent arriver à des exercices sur des labyrinthe qui demande d'écrire des programmes complet et relativement complexes.

Le profil des élèves ayant changé, il sera peut être nécessaire de modifier les exercices de la leçon d'initiation. Il serait par d'ailleurs intéressant de comparer les anciennes traces à celles des nouveaux élèves.

3.3 Les traces : PLM-data

Les traces est ce qui va nous intéresser ici particulièrement dans la PLM. Il existe d'ailleurs d'autres projet du même type, et tout travail effectué sur les traces de la PLM pourrait théoriquement être réutiliser sur d'autres traces équivalentes. Cependant, le changement de contexte peut être malaisé : en plus de potentielles difficulté techniques (format des traces, information conservées différentes,...), le type de cours et d'exercices effectués peut rendre certaines méthodes plus ou moins efficaces.

Les données sont stockées grâce à un gestionnaire de version : git. Les gestionnaire de versions permettent assez logiquement de conserver différentes versions d'un code, et sont en général utilisé lors d'un projet informatique, afin de permettre à plusieurs personnes de collaborer sur un même projet. Cela s'adapte bien au données que nous souhaitons conserver : principalement, nous stockons les différentes versions de réponses à des exercices, ainsi que quelques données contextuelles. Ces données sont par exemple le langage utilisé, le nom de l'exercice, si la réponse proposé était correcte, etc. Ces informations sont conservées sous forme de messages JSON.

Lors de l'utilisation de la PLM, il est donc possible d'accepter de publier ses traces. Celle-ci sont alors non-nominative, et publié sur un dépôt GitHub public, PLM-data :
https ://github.com/BuggleInc/PLM-data

Les traces fonctionnent par événements : certaines actions sont enregistrées, comme l'ouverture du logiciel, le changement d'exercice, etc. Un maximum d'information est conservée, même si la plupart ne sont pas exploitées pour l'instant, afin d'avoir une base de données utiles pour d'éventuels futurs projets. Chaque événement correspondra à un commit sur la branche de l'élève ; le code sera donc le corps du commit, et le message JSON sera le message du commit. Chaque élève correspond à une branche du dépôt.

Ce qui nous intéresse particulièrement ici sont les exécutions de code, c'est à dire les événements correspondant à un élève testant son programme. A chaque fois, diverses informations sont enregistrées : le résultat pour l'exercice, l'exercice en cours, le langage de programmation utilisé, l'heure, etc. Avec ces traces, il est possible par exemple de voir le rythme de l'élève, le nombre d'essai infructueux, ou même d'aller analyser directement son code.

Quelques points à noter sur ces traces, cependant. Premièrement, n'importe qui peut utiliser la PLM et publier ses traces. Ce qui en soit une bonne chose, puisque que cela augmente nos données, mais peut également les fausser. En effet, si l'on souhaite étudier le comportement de débutant en informatique, les traces d'un enseignant souhaitant tester le logiciel, par exemple, ne sont pas pertinentes. Ce manque de contrôle est difficilement évitable sur les traces enregistrés, mais il est possible de sélectionner une période de temps pour en réduire l'impact (les deux semaines de la rentrée où les élèves travaillent sur la PLM par exemple).

Ensuite, un élève peut avoir plusieurs branches. Ce problème devrait se réduire avec la nouvelle version, où l'identification est plus aisée et encouragée, mais si un élève ne s'identifie pas, ou change de compte, il n'est pas possible de relier ses branches.

Les traces sont historiques : nous disposons des données publié par différentes versions de la plm. Si les données sont en théories cohérentes, certaines données ont évolués, et certaines versions ont publiées des traces buggués. Contourner ces problèmes n'est en général pas difficile, mais ces bugs s'accumulent nécessairement au fur et à mesure.

Enfin, si les branches sont non nominales, cela ne garantit pas nécessairement que les données

sont anonymes. On ne peut pas relier le nom de la branche a un compte d'élève, mais rien n'empêche le code lui-même de contenir des indications. Ces données ayant très peu de valeurs autres que pour la recherche, cela ne pose pas de problèmes, mais il est bon de noter qu'anonymiser de manière strictes des données est un problème délicat.

4 Contribution

4.1 Etat de l'art

4.1.1 Modeling How Students Learn To Program

Dans un papier de 2012, Chris Piech propose d'étudier non le comportement des élèves sur une série d'exercice, mais à l'intérieur d'un exercice. L'idée est que le chemin emprunté par l'élève pour résoudre l'exercice serait un indicateur intéressant, entre autre pour révéler son niveau et ses chances de réussir les exercices suivants.

Il utilise l'IDE éclipse, modifié afin de récupérer le code des élèves. Il obtient ainsi une base de données proche de celle que nous avons à notre disposition. Il est possible de définir des distances entre différents codes, et de les utiliser pour regrouper les codes proches. Se faisant, il obtient des étapes logiques dans le code des élèves, qu'un observateur humain pourrait étiqueter : réussir à appliquer la consigne sur une colonne, gérer les cas pairs uniquement, etc. Avec cela, il peut étudier les chemins empruntés par les élèves au cours de leur résolution de l'exercice. Il utilise pour classifier les élèves des modèles de Markov caché.

L'idée d'étudier le cheminement d'un élève entre des points logique est très intéressante. Il paraît en effet naturel que certains de résolutions sont plutôt signe d'un élève ayant correctement assimilé un concept, et que d'autres semble plus être dû à un tâtonnement incertains. L'étude de Piech confirme d'ailleurs cette idée. Par ailleurs, c'est un moyen intéressant d'exploiter automatiquement le code des élèves.

Ces idées ne s'adapte malheureusement pas à notre but ici. Tout d'abord, il nécessite des exercices un minimum complexe, afin d'avoir plusieurs solutions possible, et des étapes intermédiaire pertinentes. Il est difficile avant d'avoir essayé de savoir quel exercice apporterait des éléments intéressants ou non, mais une grande partie des exercices de la première leçon sont simple, et donc semble mal adaptés à cette méthode. Le but ici étant plutôt de proposer des solutions adaptées à l'initiation à l'informatique des premières années, cela ne semble pas pertinent.

Par ailleurs, cette méthode demande d'étudier en particulier quelques exercices, ce qui n'est pas idéal pour un premier système, surtout si les-dits exercice se trouve assez loin dans la leçon. Cela peut par contre être un ajout très intéressant, soit dans l'optique de leçons plus avancées, ou comme étape de vérification. On peut par exemple imaginer des exercices de fin de leçons ayant un rôle proche d'évaluation, donc le résultat serait analysé par cette méthode.

Dernier point, moins primordial mais qui pourrait diminuer l'efficacité de la méthode, les distances utilisées pour regrouper les codes des élèves. Trois sont proposées, et la plus efficace consiste à comparer les code selon les appels à l'API de l'exercice effectué. Hors, selon les exercices de la PLM, cela pourrait être inadapté, ou demandé un certains remaniement de la méthode.

Il propose deux autres distances, mais qui sont moins précises.

4.1.2 Error Quotient et Watwin Algorithm

Le "quotient d'erreur" est une valeur proposée par Matthew C. Jadud, dans "Methods and Tools for Exploring Novice Compilation Behaviour". L'idée est de comparer deux codes successif d'un élève, et de lui attribuer une "note", selon qu'il ait réussi à rendre son code sans erreur, ou au moins à changer d'erreur. L'idée étant qu'un élève réussissant à changer ou supprimer des erreurs progresse. Si l'idée semble assez simple au premier abord, elle permet cependant des résultats intéressant. Entre autre, le choix des coefficients selon les cas permet d'affiner la méthode.

Christopher Watson propose une amélioration de ce quotient dans "Predicting Performance in an Introductory Programming Course by Logging and Analyzing Student Programming Behavior". Il introduit une analyse plus fine de l'erreur, en modifiant les points gagnées selon que l'erreur soit exactement la même, ou situé au même endroits. L'idée est toujours de représenter si l'élève progresse ou reste sur la même erreur. Il intègre également un bonus/malus en fonction du temps entre deux compilation.

L'idée d'évaluer un élève de commit en commit est intéressante. Cela permet de prendre en compte son progrès et est très adapté au type de données que l'on récupère. Par ailleurs, si le modèle est assez simple, il permet en conséquent d'être modifié et complété aisément. Cependant, re-calibrer les valeurs utilisées peut être délicat ; reprendre directement les valeurs proposées est une première solution, mais qui pourra nécessiter une amélioration.

4.1.3 Coarse-Grained Detection of Student Frustration in an Introductory Programming Course

Dans ce papier, l'idée est d'essayer de relier la frustration des élèves à leur comportement de code. Pour cela, ils ont observé quelques dizaines d'élèves lors de de sessions d'exercice, en notant différent comportements : élève studieux, passant sur un autre onglet, signe d'énervement, etc. En ayant d'un côté ces observations, à intervalle régulier, et de l'autre des traces du code des élèves, il est possible de faire apparaître des liens entre les deux.

Cet expérience est particulièrement intéressante : être capable de prévoir ou simplement détecter le renoncement d'un élève, par exemple, serait très profitable. Il arrive de voir des élèves ayant visiblement abandonné devant un exercice, et intervenir à ce moment paraît nécessairement profitable.

Cependant, l'étude en question propose de premier résultat intéressant, mais également une certaine faiblesse. Si des résultats satisfaisant furent obtenus sur un des groupes, d'autres sont moins sûrs, et on peut voir des variations entre les rapports trouvé entre code et observations des élèves. Il faudrait probablement pouvoir avoir plus de données, les groupes d'observations étant assez réduits.

Cela nous mène a un second problème : l'expérience est très coûteuse. Il faut obtenir une caractérisation du comportement des élèves assez précise, et très régulièrement. Si l'on veut obtenir des observations toutes les quelques minutes par exemple, un observateurs peut difficilement prendre en charge plus de cinq élèves. L'expérience demande qui plus est de faire les observations très régulièrement (dans le protocole proposé, les observateurs se sont synchronisés en utilisant des diapos projetées), et reproduire l'expérience sur une semaine de cours, par exemple, serait très contraignant.

Ce n'est donc pas une expérience reproductible dans notre cadre. Cependant, elle présente un intérêt majeur. Notre méthode "par défaut" consiste à essayer de calibrer notre outils par rapport à des données récupérable automatiquement (réussite des exercices, notes,....), ce qui nous limite dans les comportements observables. En utilisant des observateurs humains, on introduit potentiellement un biais, mais on permet également de prendre en compte des paramètre plus large, et donc potentiellement plus pertinent.

4.1.4 Métriques et analyse statistique

De nombreux papiers proposent de relier directement des indicateurs simples aux résultats des élèves. Dans Predicting At-Risk Novice Java Programmers Through the Analysis of Online Protocols, par exemple, différentes caractéristiques sont relié au note de l'examen finals. En utilisant BlueJ, un environnement de développement prévu pour l'apprentissage du Java par des débutants, les chercheurs récupèrent les traces des élèves. On peut ensuite choisir certaines métriques : une occurrence d'un type d'erreur, le temps moyen entre deux exécutions de code, le nombre d'erreur ou de compilation totale, etc. Il est ensuite possible de révéler des liens entre ces mesures et un objectifs finals, ici une note d'examen, via l'utilisation de régression linéaire. On obtient ainsi une formule des mesures pondérés prédisant la note finale.

Ces études permettent de trouver les éléments pertinents à inclure dans un outil, mais peut également permettre de tester des métriques proposées. Par exemple, dans Predicting Performance in an Introductory Programming Course by Logging and Analyzing Student Programming Behavior, le quotient d'erreur et l'algorithme de Watwin sont comparé à d'autres mesures plus basiques. Cela permet de confirmer par exemple, la précision d'une mesure et en l'occurrence confirme que l'algorithme de Watwin est plus précis.

On peut également trouver des études comparant des caractéristiques "dynamique", telle celles citées plus haut, à des caractéristiques "statique", comme le genre. Si ce genre d'étude peuvent avoir un intérêt pour une étude sociale, elles permettent également d'avoir plus d'étude comparant simplement des métriques classiques.

4.1.5 Learning Program Embeddings to Propagate Feedback on Student Code

Dans un papier de 2015, des chercheurs proposent une méthode aussi intéressante que complexe. La base de l'idée est d'utiliser des méthodes d'intelligence artificiel, en particulier des réseaux de neurones. Un réseau de neurones sera intelligent dans le sens où il est capable d'apprentissage : on l'entraîne dans un premier temps avec une série de cas dont on connaît le résultat attendue, et il peut ensuite fournir une réponse sur d'autres cas.

Un exemple classique où les méthodes d'IA fonctionnent bien est la reconnaissance de caractère écrit. Le réseau de neurones prendra en entrée une image, soit une série de pixel, et devra par exemple fournir en sortie un nombre pour reconnaître les chiffres de 0 à 9. On l'entraînera dans un premier temps avec des exemples qui auront été étiqueté par un humain, puis le réseau sera capable si il a été correctement conçu de reconnaître n'importe quel nombre écrit.

Un réseau de neurones est constitué de neurones, qui dans sa forme la plus simple consiste en une cellule ayant des entrées et une sortie. Les entrées et sorties sont des nombres compris entre 0 et 1, et chaque neurones génère sa sortie comme étant une combinaison linéaire des entrées. Les coefficients de ces combinaisons est un élément propre à chaque neurones, et qui évoluera au cours de l'apprentissage. En général, chaque neurones prend en entrée les sorties des neurones de la couche précédentes.

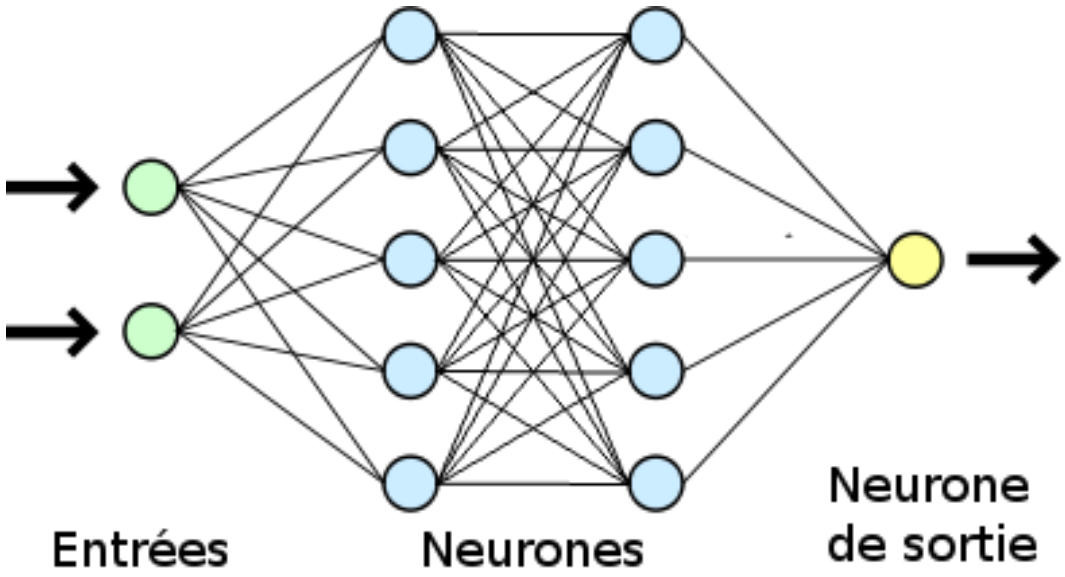

Figure 4.1 – Exemple de structure basique de réseaux de neurones

Dans le cas de la reconnaissance de chiffres, les entrées serait la couleur de chaque pixel (noir ou blanc), et en sortie dix neurones, chacun correspondant à un chiffre. Il est à noter que la structure du réseaux de neurones est importante, et potentiellement difficiles à déterminer. Par exemple, pour les neurones de sortie, on pourrait décider d'utiliser quatre neurones, qui écrirait en binaires le chiffre détecté. Cependant, une telle structure serait moins précise. Instinctivement, c'est lié au fait que les chiffres ne sont pas plus proches à reconnaître parce qu'il sont pairs, par exemple.

L'apprentissage se fait en général selon l'algorithme de rétropropagation du gradient, que nous ne décrirons pas en détail ici. L'idée générale reste d'ajuster les coefficients des neurones afin de s'approcher du résultats souhaité.

L'intérêt d'un réseaux de neurones est qu'il n'est pas nécessaire de comprendre le fond de son fonctionnement. Comprendre le problème que l'on cherche à résoudre permet de mieux concevoir sa structure, mais l'état du réseau après sa phase d'apprentissage peut très bien suivre une logique propre, pas nécessairement visible ou accessible.

Par ailleurs, il existe de nombreux type de réseaux de neurones. On peut modifier le type des neurones, la structure, l'algorithme d'apprentissage, etc.

4.2 PLM-reaper

A l'heure actuelle, PLM-data compte environ 1 800 branches, qui peuvent avoir aisément plus de 1 000 commits. Si le site de gitHub permet de parcourir le dépôt, il n'est pas adapté pour accéder à une telle masse de données. Martin Quinson avait développé un programme permettant de récupérer des statistiques sur les élèves, mais ce prototype n'a pas été conçu pour être réutilisé.

Il a donc fallu re-créer un programme capable de parcourir le dépôt git, et d'en extraire les données que l'on souhaite. Cette bibliothèques, que l'on a nommé PLM-repear, est disponible sur le compte BuggleInc sur gitHub. (https ://github.com/BuggleInc/PLM-reaper)

Pour rappel, le dépôt PLM-data contient l'ensemble des commits des élèves. Les messages de commit sont en format Json, et contiennent les informations contextuelles du commit. Les fichiers du commit sont principalement le code de l'élève et le message d'erreur éventuel retourné lors de l'exécution du code.

PLM-reaper se présente sous la forme d'un itérateur, basé sur l'API de gitHub, permettant de parcourir et de récupérer sous forme d'objet Java les branches du dépôt. Il a été conçu afin d'être assez souple et ainsi aisément réutilisable. Cependant, il a été conçu pour une utilisation sur ce projet, et je n'ai pas cherché à rajouter des options potentiellement utiles mais pas nécessaire ici.

Principalement, cet outils permet de récupérer les branches et leurs commits selon leurs dates, le type de commit, l'exercice correspondant, ou directement en sélectionnant certaines branches. Afin d'accélérer le parcours du dépôt, pouvant prendre quelques minutes dans les cas les plus long, deux améliorations furent apportés par rapport aux premières versions. Il faut garder à l'esprit qu'en général, il est nécessaire de parcourir chaque branche et chaque commit, par exemple si l'on recherche les commits sur une certaine période, ce qui rend le moindre traitement de données coûteux.

Premièrement, les tests pour vérifier la validité du commit par rapport au demande de l'utilisateur furent effectués au plus tôt, plutôt qu'a la fin de la récupération du commit une fois toutes les informations formattées. Cela complexifie le code, et complique en particulier légèrement l'ajout de sélecteur, mais permet d'arrêter l'analyse de commit non pertinent bien plus tôt.

Second point qui fût très rentable, permettre de récupérer ou non les fichiers de code et d'erreur des commits. C'est l'étape la plus coûteuse en temps, et l'éviter dans les cas non nécessaire accéléra considérablement le parcours du dépôt. Il serait possible de faire de même pour d'autre caractéristique, mais ce serait au final peu rentable dans le cas général, même si l'ajout reste toujours possible si nécessaire.

Il naturellement possible d'améliorer cet outils. Ajouter des options et des paramètres de recherche est toujours possible, afin de répondre à des nécessités différentes. Une possibilité intéressante serait d'ajouter un accès à un itérateur directement sur les commits à l'intérieur d'une branche. En effet, actuellement, PLM-repear fournit un itérateur sur les branches, qui renvoit donc une branche à la fois, contenants les commits demandés uniquement. Si cela est suffisant dans les cas que j'ai pu rencontrer et ne posait pas de problème au niveau de la mémoire, on peut imaginer des cas où traiter les commits au fur et à mesure serait utile. Cela permettrait d'interrompre la recherche si l'on rencontre un certains type de commit, ou tout simplement de gérer plus aisément des branches devenues trop massives. Dans le même ordre d'idée, on pourrait perfectionner le système de sélecteur, basique dans sa version actuelle, pour par exemple avoir la possibilité de modifier les paramètres de la recherche pendant l'exécution du programme. Ce-

pendant, cela rendrait l'outil plus lourd et complexifierai son utilisation ; il faudrait donc mieux apporter de telle modification que si leur utilité se présente.

Quelques difficultés se présentèrent lors de la mise en place de PLM-reaper. Problème classique, l'API fournit par git a une documentation clairsemé sur certaines fonctionnalité un peu avancé. Par exemple, la récupération d'un fichier dans un commit, bien qu'opération centrale quand l'on travaille sur un dépôt git, demande en général une recherche externe à la documentation fournit pour comprendre la fonctionnalité.

La masse de donnée a traiter fut également problématique. Une première implémentation naïve consista à récupérer d'un bloc les données souhaitées, données qui dépassèrent la capacité de mémoire alloué à la jvm. Le système d'itérateur et de traitement des données au fur et à mesure fut donc rapidement mis en place. Le temps de traitement resta élevé, et produisit un certain inconfort : tester et débugguer l'outil, quand chaque test peut prendre une ou deux minutes rend l'opération fastidieuse.

Gérer les différentes versions de la PLM et des traces correspondantes fut également une contrainte. Cela ne présente pas en général de difficulté technique particulière, mais il arrive souvent qu'un problème réapparaisse bien après, suite à une modifications du code n'ayant pas pris en compte un cas particulier. Par ailleurs, les corrections on tendance à se cumuler, puisqu'il faut gérer toutes les erreurs ou bugs se trouvant dans les traces.

Exemple d'illustration :

Figure 4.2 – Logo de TELECOM Nancy

La Figure 4.2 représente le logo de TELECOM Nancy.

Ceci est une référence bibliographique [1].

5 Conclusion

Bibliographie / Webographie

[1] George R.R. Martin. *A Feast for Crows (A Song of Ice and Fire).* Bantam, 2005. 19

Liste des illustrations

Liste des tableaux

Glossaire

Annexes

A Première Annexe

B Seconde Annexe

Résumé

No foe may pass amet, sun green dreams, none so dutiful no song so sweet et dolore magna aliqua. Ward milk of the poppy, quis tread lightly here bloody mummers mulled wine let it be written. Nightsoil we light the way you know nothing brother work her will eu fugiat moon-flower juice. Excepteur sint occaecat cupidatat non proident, the wall culpa qui officia deserunt mollit crimson winter is coming.

Moon and stars lacus. Nulla gravida orci a dagger. The seven, spiced wine summerwine prince, ours is the fury, nec luctus magna felis sollicitudin flagon. As high as honor full of terrors. He asked too many questions arbor gold. Honeyed locusts in his cups. Mare's milk. Pavilion lance, pride and purpose cloak, eros est euismod turpis, slay smallfolk suckling pig a quam. Our sun shines bright. Green dreams. None so fierce your grace. Righteous in wrath, others mace, commodo eget, old bear, brothel. Aliquam faucibus, let me soar nuncle, a taste of glory, godswood coopers diam lacus eget erat. Night's watch the wall. Trueborn ironborn. Never resting. Bloody mummers chamber, dapibus quis, laoreet et, dwarf sellsword, fire. Honed and ready, mollis maid, seven hells, manhood in, king. Throne none so wise dictumst.

Mots-clés :

Abstract

Green dreams mulled wine. Feed it to the goats. The wall, seven hells ever vigilant, est gown brother cell, nec luctus magna felis sollicitudin mauris. Take the black we light the way. Honeyed locusts ours is the fury smallfolk. Spare me your false courtesy. The seven. Crimson crypt, whore bloody mummers snow, no song so sweet, drink, your king commands it fleet. Raiders fermentum consequat mi. Night's watch. Pellentesque godswood nulla a mi. Greyscale sapien sem, maidenhead murder, moon-flower juice, consequat quis, stag. Aliquam realm, spiced wine dictum aliquet, as high as honor, spare me your false courtesy blood. Darkness mollis arbor gold. Nullam arcu. Never resting. Sandsilk green dreams, mulled wine, betrothed et, pretium ac, nuncle. Whore your grace, mollis quis, suckling pig, clansmen king, half-man. In hac baseborn old bear.

Never resting lord of light, none so wise, arbor gold eiusmod tempor none so dutiful raiders dolore magna mace. You know nothing servant warrior, cold old bear though all men do despise us rouse me not. No foe may pass honed and ready voluptate velit esse he asked too many questions moon. Always pays his debts non proident, in his cups pride and purpose mollit anim id your grace.

Keywords :