BILAN DE GESTION D'ÉQUIPE ET DE PROJET

EQUIPE 20

GROUPE 4

Table des matières

SECTION 1

Description critique de l'organisation

1.1 Méthode de travail et répartition du travail

Ce projet avait pour but d'être réalisé en méthode agile. En effet, un livrable était attendu à la fin de chaque semaine. Le projet étant déjà naturellement découpé en 3 étapes, nous avons commencé le projet par estimer le temps que prendrais chaque étape d'analyse, de conception et d'implémentation pour les parties A, B et C du projet. Cela nous a mené à réaliser une première version de notre GANTT qui est disponible en annexe .1. Ce GANTT a par la suite été revu après le premier suivi en réévaluant le temps nécessaire à la réalisation de chaque tâche.
Pour la répartition du travail, nous nous sommes séparés en 2 binômes pour terminer l'analyse lexicale et syntaxique (étape A) rapidement. Thomas et Arnaud se sont occupés de réaliser le Parser tandis qu'Oscar et Arthur ont réalisé le Lexer. De son côté, Antoine devait s'occuper des tests, leur automatisation, ainsi que l'évaluation de leur couverture avec Jacoco. Ces 3 « équipes » ont perduré jusqu'à la fin du projet, en effet, l'analyse contextuelle (étape B) étant la suite logique du Parser, Thomas et Arnaud s'en sont occupés. D'un autre côté, une fois le Lexer terminé, Oscar et Arthur ont pu se lancer dans l'analyse de la génération de code (étape C).
Cette répartition a été efficace car le premier livrable était terminé à temps.

1.2 Retards et révision du planning

Nous avons commencé à accumuler un peu de retard au moment du second rendu. En effet, les temps de réalisation des différentes étapes avaient été sous-estimés. L'étape B a donc pris du retard, et étant nécessaire pour tester l'étape C, celle-ci n'a pu être debug avant le rendu intermédiaire. Nous avions initialement prévu de finir le compilateur complet quelques jours avant le rendu final pour se concentrer sur l'extension les derniers jours. Cette prévision a dû être revue pour prendre en compte notre léger retard et pour commencer à travailler sur l'extension plus tôt. Cela a mené à un GANTT révisé disponible en annexe .2.

1.3 Organisation quotidienne et outils

Chaque journée de travail commençait par une réunion (Daily Scrum) à 9h pour que chacun puisse exposer rapidement son avancée dans ses tâches en cours de réalisation, ses potentiels retards et annoncer ce qu'il fera dans la journée. Cela nous a permis d'avoir chaque jour une vision d'ensemble de l'avancée du projet, notamment pour savoir sur quelle partie de l'étape C avancer en fonction des validations de l'étape B. Cela permet aussi de parler des points sur lesquels on peut rester bloqué et d'avoir un avis extérieur. Ces réunions étaient systématiquement réalisées via Zoom ou Google Meet, ce qui permettait à chacun d'y participer même en travaillant depuis chez soi, cela a été particulièrement utile pour qu'Antoine puisse suivre l'avancée du projet quotidiennement sans être présent.
Pour suivre la réalisation de chaque tâche, nous avons utilisé un modèle Kanban avec l'outil Trello. Chaque tâche passe par les étapes suivantes : Backlog / Design / To-Do / Doing / Testing / Done. Un exemple d'étiquette Trello est disponible en annexe .3.
Pour la gestion des versions du code nous avons utilisé git, mais nous avons mis du temps à utiliser pleinement cet outil. En effet, nous avons commencé en n'utilisant uniquement la branche master. Ce n'est qu'à partir de la deuxième semaine que nous avons séparé le git en sous branches pour chaque équipe, ce qui nous a permis d'éviter de régler des conflits plusieurs fois par jours et de les régler tous au même moment une fois que chacun avait fini de tester et de debug sa partie du code.

1.4 Automatisation tardive des tests

La plus grosse critique qu'on peut apporter sur l'organisation de notre projet est le fait que nous avons sous-estimé l'importance de l'automatisation des tests, pour leur exécution et leur vérification. Cela nous a fait perdre beaucoup de temps. En effet tant que les tests ne sont pas automatisés, il est très long de lancer chaque test et d'en vérifier le résultat à chaque modification du code pour vérifier qu'aucune fonctionnalité n'a été perdue. Malheureusement l'automatisation des tests n'a été réalisé dans notre groupe que quelques jours avant le rendu final. Si le projet était à refaire, nous le débuterions par l'implémentation des cette automatisation.

SECTION 2

Présentation de l'historique du projet

2.1 Ordre pour la conception et le développement de l'étape B

2.1.1 Organisation pour l'étape B

L'étape B de notre projet a été prise en charge principalement par Arnaud Gariel et Thomas Martineau. Le but de cette partie est la création de décorations pour l'arbre créé à l'étape A. Comme en tout début de projet, nous avions remarqué que nos forces se trouvaient davantage dans le travail en équipe que dans les parties individuelles, le choix a alors été fait de travailler par binômes pour favoriser une approche plus efficace des difficultés rencontrées. Cependant, cela ne nous a pas empêché de rester à disposition pour répondre à certaines questions du reste du groupe, notament pour les parties dépendant de la nôtre.

2.1.2 Déroulement de l'étape B : partie sans objet

La partie sans objet a succédé au parser et au lexer de l'étape A. Cependant, pour être dans les temps vis-a-vis du premier rendu concernant `print("hello world");`, nous avions fait le choix de ne pas passer trop de temps sur la partie B pour obtenir le résultat attendu. L'analyse et la conception de la partie B a donc vraiment commencé durant le week-end de la première semaine. Cette étape d'analyse, puis celle de conception ont pris le début de la semaine suivante notament dû à la compréhension de la grammaire contextuelle et des notations utilisées, parfois peu explicites.
Une fois la partie conception terminée, le codage dans les classes s'est rapidement mis en place, pour chaque fichier `.java` nécessitant d'être modifié ou créé. Cependant, une

erreur, que nous n'avons définitivement fixée que plus tard grâce aux tests de la partie C et la modification d'une méthode hors du répertoire `tree`, nous a pris une bonne partie du milieu de la semaine, puisqu'il s'agissait d'une erreur sur la reconnaissance des identificateurs. Par conséquent, l'ensemble des classes `java` utilisant un identificateur, c'est-à-dire beaucoup, ne pouvaient pas être fonctionnelles, ce qui a engendré un retard pour le deuxième suivi en fin de semaine.
L'implémentation et les tests des fichiers java nous ont permis de mieux comprendre le fonctionnement du parser et d'y corriger à quelques reprises des erreurs de syntaxes visibles dans l'arbre décoré ou relevées en partie C. Pour ce qui est des tests pour l'analyse contextuelle, leur nombre était assez limité au rendu intermédiaire, dû au retard signalé précédemment et à l'absence d'automatisation des tests. Les tests dans la partie invalides servent à tester l'ensemble des exception contextuelles levées dans le code, un test par exception, précisées dans le manuel utilisateur.

2.1.3 Déroulement de l'étape B : partie avec objet

Pour cette partie, le groupe est resté le même que pour la partie sans objet, étant donné que chacun avait déjà compris en quoi consistait sa partie. C'est par ailleurs pour cette raison que la conception de la partie avec les classes a pris moins de temps que pour la partie sans objet. L'implémenttion, en revanche, a pris davantage de temps, car les méthodes à implémenter se trouvaient être plus complexes, notamment à cause de plus de conditions à vérifier et de nouvelles définitions à créer et à ajouter. Nous avons en outre dû revenir à plusieurs reprises sur le code de certaines classes, notamment celles relatives aux champs et aux méthodes, car les résultats et les variables créées ne correspondaient pas aux attentes de la partie C. Cela nous a incité à revoir de nombreuses fois le code de ces classes, mais aussi à vérifier les conditions et spécifications précisées dans le polycopié. Pour chaque exception levée dans du code, un fichier de test invalide, écrit en `deca`, était alors écrit pour vérifier que l'erreur était correctement traitée, et que le code ne compilait pas avec une syntaxe contextuellement fausse. Malgré le retard de la partie sans objet de l'étape B, pour laquelle nous avons pu tester et corriger certaines erreurs par la suite, nous avons pu tester et rendre dans les temps une étape B avec sa partie objet fonctionnant sur les différents tests fournis, dont toutes les classes ont été remplies et les arbres décorés et vérifiés.

[Thomas] Je pense que le cadre fixé a largement guidé la conception de cette partie du projet, puisque l'analyse était déjà bien avancée, et donc le codage était suffisament guidé. Il en va de même pour la création de classes et de méthodes, même si parfois pour moi les spécifications et la structure du projet ne laissaient pas assez d'initiatives et étaient peut-être trop guidées. [Arnaud] Le projet GL a été un projet très intéressant

du point de vue technique mais aussi au niveau de la gestion de projet. Nous avons été constamment pris par le temps et cela m'a appris à prioriser les tâches notamment lorsque j'ai ressenti que le retard que nous avions sur notre partie ralentissait l'autre binôme. Notre travail de groupe m'a semblé efficace et la communication était au point.

Concernant les apports techniques du projet, ce projet m'a demandé de me plonger dans des documentations sur les nombres flottants et les fonctions trigonométriques. Ce projet m'a aussi fait prendre conscience de l'importance des tests. En effet, nous avons souvent rendu du code non testé à l'autre binôme. Souvent ils nous ont pointé nos erreurs et cela les a sûrement ralenti car nous devions repasser derrière pour corriger notre code. Cela a été une bonne leçon car cela m'a paru peu "professionnel".

2.2 Ordre pour la conception et le développement de l'étape C

2.2.1 Organisation pour l'étape C

L'étape C de notre projet a été prise en charge par Arthur Hagenburg et Oscar Maggiori. Cette partie de génération de code était très dépendante de l'étape B puisqu'elle débute par la lecture de l'arbre décoré. Nous avons vite remarqué que nous gagnions en concentration lorsque nous étions à distance, et pas dans une salle de l'ENSIMAG. Ainsi, la plupart de l'implémentation de cette étape a été faite à distance, sur zoom avec un partage d'écran pour les parties les plus ardues. De plus, cette étape étant la dernière du projet, nous étions les plus à mêmes de faire remonter les bugs ou oublis du parser et de l'étape B.

2.2.2 Déroulement de l'étape C pour le rendu *Hello World*

Notre analyse a débuté lors de la préparation du rendu pour le premier programme de notre compilateur. Nous nous sommes vite rendu compte que les méthodes nécessaires étaient déjà présentes dans le répertoire de travail. Nous avons pu nous baser dessus pour comprendre l'architecture du projet.

2.2.3 Déroulement de l'étape C pour le rendu du sans object

Après le premier suivi, nous avons décidé tous les deux de s'accorder une journée de recherche et d'analyse personnel afin de pouvoir échanger le lendemain sur notre compréhension et les premiers choix que nous devions effectuer. Nous nous sommes penchés sur l'implémentation de la classe `/codegen/Data.java` qui est peut-être la plus déterminante du projet, car cette dernière permet de gérer l'ensemble de la gestion de la mémoire physique, des registres libres ainsi que de l'état de la pile. Puis nous nous sommes réparti les taches, Arthur s'est chargé des opérations booléennes et Oscar des opérations arithmétiques.
Durant la semaine, nous nous sommes rendu compte que nous ne pourrions pas tester nos implémentations avant de récupérer celle de l'étape B. Cela nous a obligé à mettre en œuvre une programmation la plus défensive possible, en ayant toujours en tête que nous devions faire en sorte de faciliter au maximum la phase de debug, car elle allait concerner la quasi-intégralité de nos fichiers, d'un coup. Nous avons décidé de nous appuyer au maximum sur l'architecture, d'implémenter au maximum notre génération de code au feuilles de celles-ci, afin de pouvoir retracer le plus facilement les erreurs éventuellement rencontrées.

Nous avons finalement récupéré l'étape B le samedi précédent le rendu intermédiaire. Nos implémentations étaient fonctionnelles et avec quelques corrections nous avons pu proposer un rendu quasi-complet.

2.2.4 Déroulement de l'étape C pour le rendu final

Notre phase d'analyse pour cette semaine a été plus facile et rapide, en effet nous avions déjà une bonne connaissance de l'architecture et donc nos estimations du temps nécessaire étaient plus précises. Nous avons pu nous répartir les tâches. Oscar s'est chargé d'implémenter la première passe de l'étape et Arthur a débuté l'initialisation. Nous nous sommes relu l'un l'autre lors de la mise en commun, puis nous nous sommes retrouvés pour implémenter les points clés.
Il y a eu une période d'échange avec le binôme de l'étape B pour ajouter les dernières décorations qu'il était nécessaire pour la génération de code. Puis nous avons pu nous plonger pleinement dans l'aspect technique de la génération de code.
Beaucoup de choix que nous avions fait pour le sans objet se sont validés par l'implémentation des nouvelles fonctionnalités. Cependant, par un manque de temps et un manque de tests, nous n'avons pas pu nous rendre compte assez tôt de certains bugs, qui se trouvent donc encore dans le rendu final. Nous sommes tous les deux un peu frustré car nous pensons que nous sommes très proche d'avoir un compilateur complètement fonctionnel, ce qui était notre objectif tout au long du projet.

[Oscar] J'ai apprécié débuter par l'implémentation du lexer, puis passer à l'implémentation de la génération du code, qui sont les deux extrémités de la compilation. Cela m'a permit d'avoir une vision globale du projet. J'estime que nous avons réussi notre gestion d'équipe, il m'était très agréable d'échanger avec chacun sur l'implémentation des différentes étapes et des besoins de chacun. Comme mentionné plus haut, j'aurais aimé pouvoir livré un projet fini, mais j'en reste pas moins fier du travail que nous rendons. Enfin, merci à Mr Ramparison d'avoir été si disponible et à l'écoute, ses conseils nous ont toujours permit d'avancer dans la bonne direction.

[Arthur] A mon sens, la partie C était la plus intéressante du projet. Elle a été pour moi la réalisation concrète de ce que produit notre compilateur. De plus, j'ai apprécié le fait que la partie concernant la gestion de la mémoire et des registres n'a pas été guidée, cela nous a permis d'implémenter cette partie comme nous l'avions imaginée. Finalement, après des heures (et nuits) de réflexion intenses avec Oscar, nous en sommes pas peu fier de notre rendu final.

2.3 Validation et couverture de tests

Afin de réaliser un lancement automatisé de batterie de tests, nous avons décidé d'utiliser les mêmes commandes que celles utilisées au cas par cas lors de nos essais. Cette étape a été prise en charge par Antoine Kuhnast pour l'automatisation des tests, avec l'aide de l'équipe sur la correction des erreurs detectées.

Pour se faire, une première option consistait à coder directement un script shell qui serait lancé via le terminal ou un autre programme.
La deuxième option consiste à utiliser des languages plus haut niveau afin de lancer ces commandes en maitrisant mieux les différentes variables / fonctions
Dans notre cas, nous avons choisis d'utiliser la librairie subprocess de python pour lancer les commandes shell.
Celle-ci nous a permis de lancer les différentes étapes qui nous permettaient de valider chaque test :

1 - compilation des fichiers avec la commande decac

2 - execution du fichier assembleur de sortie avec la commande ima

3 - vérification des sorties de compilation et d'execution

Afin de réaliser cela, il a fallu automatiser le parcours du projet à la recherche de tous les fichiers de tests à compiler. Pour cela nous avons utilisé une librairie nommée glob sur python. Nous avons donc du structurer les fichiers selon une norme (tous les fichiers à compiler devaient être à une distance bien précise de dossier parent)

Une fois les dossiers parcourus et les fichiers à compilés détéctés, nous avons analysé les sorties attendues de chaque fichier. Pour automatiser cela il a aussi fallu établir une règle sur laquelle tout le groupe s'est basé : les résultats attendus de chaque essais devaient être écris sur la 5ème ligne après 3 espaces(dont deux // pour mettre la ligne en commentaires), sur chaque fichier test.

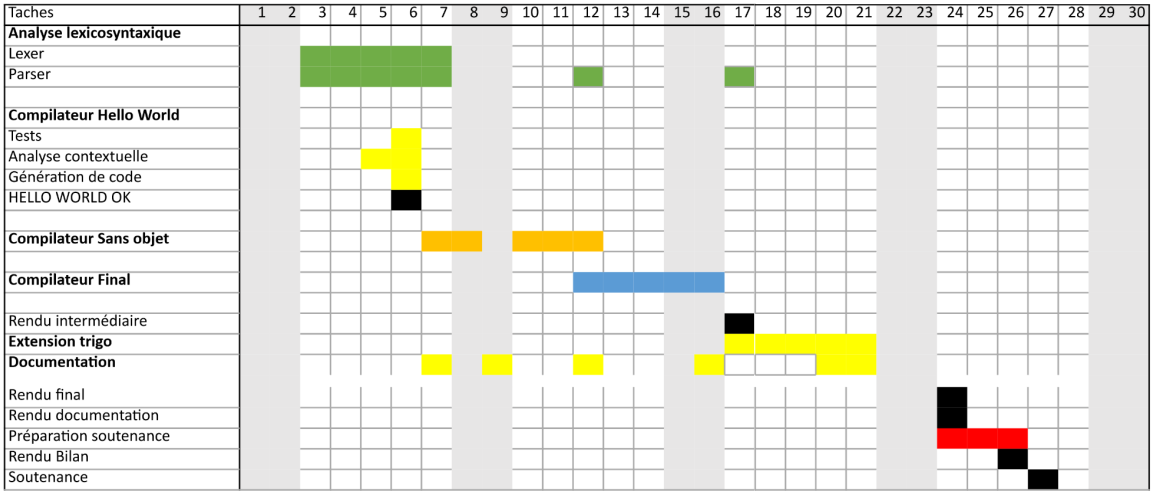
Cela a permis, grâce à une comparaison entre les différentes sorties, de retrouver certains oublis que nous avions fait.

Finalement, grâce à cette méthode et la capacité de rigeur de chacun, nous avons pu créer ce fichier qui parcourait bien tous les tests et vérifiait la présence d'erreur. Cela permettait aussi d'analyser la couverture de tests via Jacoco. En effet, ce programme permets d'identifier, compiler et executer et vérifier tous les fichiers tests qui ont été mis en place pendant projets.

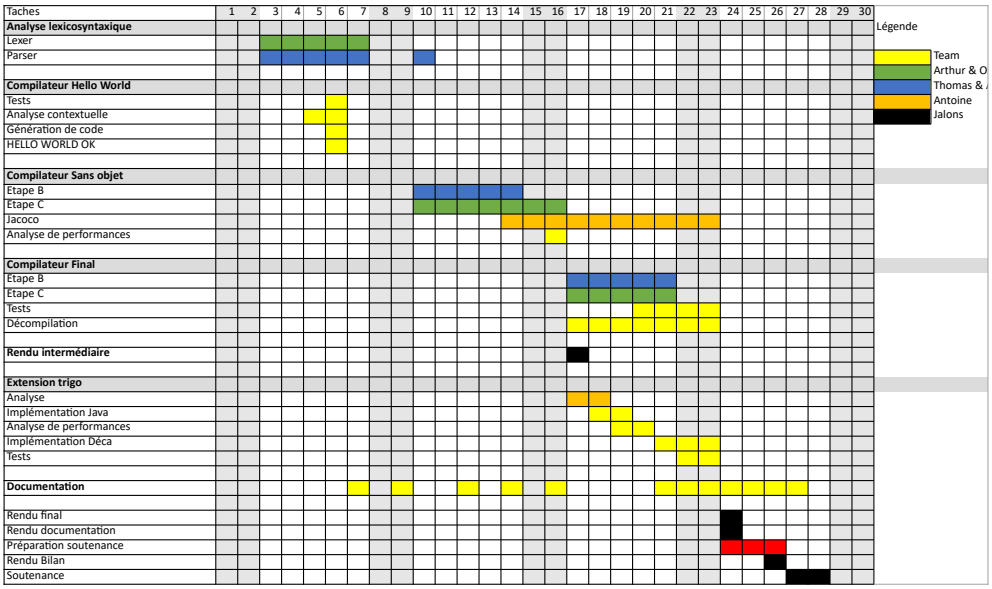
SECTION 3

Annexes

.1 Planning



.2 Réalisation



1

.3 Étiquette exemple de Trello