



TELECOM Nancy

PCL1

---

## Projet Compilation - Langage Blood

---

Membres projet :  
Marie-Astrid Chanteloup  
Lucille Delaporte  
Bleunwenn Graindorge  
Niels Tilch

Responsables de module :  
Suzanne Collin  
Gerald Oster





# Table des matières

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Fiche Projet . . . . .	5
1.1.1	Contexte du projet . . . . .	5
1.1.2	Objectif et intérêt du projet . . . . .	5
1.1.3	Rédacteurs, commanditaires . . . . .	5
1.1.4	Présentation générale du projet . . . . .	5
1.1.5	Livrables . . . . .	5
1.1.6	Date limite . . . . .	6
1.2	Organisation du document . . . . .	6
1.3	Précisions légales . . . . .	6
1.4	Clause de non-plagiat . . . . .	6
<b>2</b>	<b>Gestion de projet</b>	<b>11</b>
2.1	Équipe de projet . . . . .	11
2.2	Analyse du projet . . . . .	11
2.2.1	Définition des objectifs . . . . .	11
2.2.2	Analyse des risques : Matrice SWOT . . . . .	11
2.3	Organisation du projet . . . . .	12
2.4	Outils de travail . . . . .	13
2.4.1	Réunion . . . . .	13
2.4.2	Partage du travail . . . . .	13
2.4.3	Rédaction du rapport . . . . .	13
2.5	Comptes-rendu des réunions . . . . .	13
2.5.1	Réunion du 14 octobre 2020 . . . . .	13
2.5.2	Réunion du 21 octobre 2020 . . . . .	15
2.5.3	Réunion du 18 novembre 2020 . . . . .	16
2.5.4	Réunion du 2 décembre 2020 . . . . .	16
2.5.5	Réunion du 9 décembre 2020 . . . . .	17
2.5.6	Réunion du 11 janvier 2021 . . . . .	18
<b>3</b>	<b>Grammaire du Langage</b>	<b>21</b>
3.1	Parties de la grammaire . . . . .	21
3.2	Déclaration d'une classe . . . . .	21
3.2.1	La classe . . . . .	21
3.2.2	Les paramètres . . . . .	21
3.2.3	L'héritage . . . . .	21
3.2.4	Bloc de classe . . . . .	22
3.2.5	Constructeur et méthodes . . . . .	22
3.3	Déclaration d'un attribut . . . . .	22
3.4	Déclaration d'une méthode . . . . .	22
3.5	Expressions et instructions . . . . .	23
3.5.1	Règles sur les instructions . . . . .	23
3.5.2	Règles sur les expressions . . . . .	23
3.5.3	Les règles sur les boucles . . . . .	23
3.5.4	Les règles sur les opérations et feuilles terminales . . . . .	23
3.6	Limites de la grammaire . . . . .	24
<b>4</b>	<b>Structure de l'arbre abstrait</b>	<b>25</b>
4.1	Base d'un programme . . . . .	25
4.2	Déclaration d'une classe . . . . .	25
4.3	Déclaration d'un attribut et d'une méthode . . . . .	25
4.4	Instruction et expression . . . . .	25

- 5 Tests 27
  - 5.1 Expressions arithmétiques, affectation et déclarations de variables 27
    - 5.1.1 Déclaration de variables 27
    - 5.1.2 Opérations arithmétiques 27
    - 5.1.3 Expressions conditionnelles 28
    - 5.1.4 Erreur de syntaxe soulevée par la grammaire 28
  - 5.2 Boucles conditionnelles (if et while) 28
    - 5.2.1 Boucles conditionnelles simples 28
    - 5.2.2 Les boucles imbriquées 29
    - 5.2.3 Erreurs de syntaxe soulevées par la grammaire 29
  - 5.3 Définitions de classe, constructeur méthodes et paramètres 30
    - 5.3.1 Déclaration de classe 30
    - 5.3.2 Déclaration de méthode 30
    - 5.3.3 Erreurs de syntaxe évitées par la grammaire 30
  - 5.4 Appels de méthodes 30
    - 5.4.1 Les appels de méthode successifs et leurs arguments 30
    - 5.4.2 Erreurs de syntaxe évitées par la grammaire 31
- 6 Bilan du projet 33
  - 6.1 Rapport d'étonnement 1 - Novembre 2020 33
    - 6.1.1 Marie-Astrid CHANTELOUP 33
    - 6.1.2 Lucille DELAPORTE 33
    - 6.1.3 Bleunwenn GRAINDORGE 33
    - 6.1.4 Niels TILCH 33
  - 6.2 Bilan de la première partie du projet 34
  - 6.3 Bilan du projet par membre 34
    - 6.3.1 Marie-Astrid CHANTELOUP 34
    - 6.3.2 Lucille DELAPORTE 34
    - 6.3.3 Bleunwenn GRAINDORGE 35
    - 6.3.4 Niels TILCH 35
  - 6.4 Travail réalisé 35

# Chapitre 1

## Introduction

### 1.1 Fiche Projet

#### 1.1.1 Contexte du projet

Ce projet a été réalisé dans le cadre des modules PCL1 et PCL2 de la deuxième année du cycle ingénieur sous statut étudiant de TELECOM Nancy.

L'objectif est de concevoir un compilateur pour le langage Blood

Ce projet est scindé en 2 parties. La première concerne la construction de la grammaire et de l'AST (Abstract Syntax Tree, ou arbre syntaxique abstrait en français) associés au langage. La seconde partie concerne la construction de la table des symboles et le développement du compilateur final du langage Blood.

#### 1.1.2 Objectif et intérêt du projet

Le projet de Compilation permet de mettre en pratique et d'approfondir les compétences acquises en cours de Traduction des Langages. L'objectif de ce projet est de réaliser un compilateur pour un langage orienté objet. Il permet également de mettre en application les cours de PFSI de première année à TELECOM Nancy, car le compilateur produira en sortie du code assembleur.

Mais la PCL est également une occasion de mettre en pratique les enseignements du cours de MO de première année, à travers la gestion de projet à fournir, et de travailler sur l'utilisation d'un dépôt Git.

#### 1.1.3 Rédacteurs, commanditaires

Ce document a été rédigé par CHANTELOUP Marie-Astrid, DELAPORTE Lucille, GRAINDORGE Bleunwenn et TILCH Niels, les quatre membres de l'équipe projet. Le projet quant à lui a été commandité par les membres de l'équipe pédagogique du projet Compil qui est au nombre de 2 : COLLIN Suzanne, OSTER Gerald.

#### 1.1.4 Présentation générale du projet

Le sujet du projet Compil s'inspire d'un travail proposé aux étudiants de l'Université Paris-Saclay. L'objectif est d'écrire le compilateur du langage Blood. Le travail sera décomposé en deux grandes parties. Pour la première partie, appelée PCL1, il sera nécessaire d'utiliser l'outil ANTLR afin de réaliser la définition complète d'une grammaire et un arbre abstrait. Il faudra s'assurer que la grammaire est bien LL(1) et la tester sur des exemples de programmes.

Lors de la deuxième partie du projet, dite PCL2, il faudra contruire la table des symboles, réaliser les contrôles sémantiques et générer le code assembleur correspondant. Le code généré devra être en langage d'assemblage *microPIUP/ASM*.

#### 1.1.5 Livrables

Ce projet étant évalué, l'équipe projet devra rendre un certain nombre de livrables, voici la liste exhaustive de ceux de la première partie du module :

- La grammaire du langage
- La structure de l'arbre abstrait
- Des jeux d'essais permettant d'illustrer le fonctionnement et les limites de la grammaire et de l'arbre abstrait
- Les outils de gestion de projet

Le tout devra être rendu dans un rapport rédigé par l'ensemble du groupe.

Les livrables du module de PCL2 sont les suivants :

- La structure de la table des symboles
- Les erreurs sémantiques reconnues et traitées par le compilateur
- Des schémas de traduction vers le langage assembleur
- Des jeux d'essais supplémentaires
- Les outils de gestion de projet

### 1.1.6 Date limite

Le projet étant divisé en deux parties, il y aura deux rapports à rendre. Un premier à la fin du module PCL1 pour le 15 janvier 2021. La fin du module PCL2, correspondant à la fin du projet, est fixée au 16 avril 2021.

## 1.2 Organisation du document

Ce document rapporte la première partie du projet.

Dans le chapitre 2, nous présentons les éléments ainsi que les outils de gestion de projet que nous avons utilisés.

Dans le chapitre 3, nous présentons la conception de la grammaire du langage Blood.

Dans le chapitre 4, nous présentons la structure de l'AST au travers de ses tokens.

Dans le chapitre 5, nous présentons les tests que nous avons réalisés pour vérifier le bon fonctionnement de la grammaire et les limites que nous avons dénotée à ce moment-là.

Dans le chapitre 6, nous avons consigné les rapports d'étonnement ainsi que le bilan de la première partie du projet, d'un point de vue personnel et global.

## 1.3 Précisions légales

Ce projet n'est pas destiné à un usage commercial, ainsi, les images présentes, notamment les images de test, ne sont pas destinées à la publication et ne sont pas toutes libre de droit.

Cependant, le caractère strictement scolaire de ce projet nous autorise à les inclure en accord avec :

- Code civil : articles 7 à 15, article 9 : respect de la vie privée
- Code pénal : articles 226-1 à 226-7 : atteinte à la vie privée
- Code de procédure civile : articles 484 à 492-1 : procédure de référé
- Loi n78-17 du 6 janvier 1978 : Informatique et libertés, Article 38

## 1.4 Clause de non-plagiat

Les déclarations sur l'honneur de non-plagiat des membres de l'équipe projet sont présentes dans les quatre pages suivantes.

## Déclaration sur l'honneur de non-plagiat

**Je, soussigné(e),**

**Nom, prénom :** Chanteloup Marie-Astrid

**Élève ingénieur(e) régulièrement inscrit en 2<sup>me</sup> année à TELECOM Nancy**

**Numéro de carte étudiante :** 31916886

**Année universitaire :** 2020/2021

**Auteur, en collaboration avec Delaporte Lucille, Graindorge Bleunwenn et Tilch Niels du rapport :**

### Projet Compilation - Langage Blood

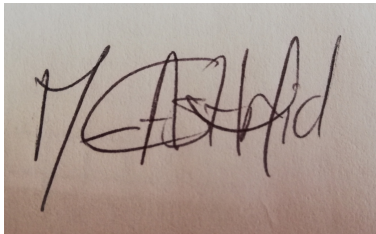
Par la présente, je déclare m'être informé(e) sur les différentes formes de plagiat existantes et sur les techniques et normes de citation et référence.

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 Nancy, le  
Signature : 14/01/2020

A photograph of a handwritten signature in dark ink on a light-colored surface. The signature is stylized and appears to read 'M. Chanteloup'.

## Déclaration sur l'honneur de non-plagiat

**Je, soussigné(e),**

**Nom, prénom :** Delaporte, Lucille

**Élève ingénieur(e) régulièrement inscrit en 2<sup>me</sup> année à TELECOM Nancy**

**Numéro de carte étudiante :** 0710001189Y

**Année universitaire :** 2020/2021

**Auteur, en collaboration avec Chanteloup Marie-Astrid, Graindorge Bleunwenn et Tilch Niels du rapport :**

### Projet Compilation - Langage Blood

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.

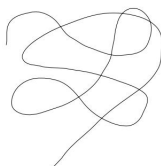
Je déclare en outre que le travail rendu est un travail original, issu de ma réflexion personnelle, et qu'il a été rédigé entièrement par mes soins. J'affirme n'avoir ni contrefait, ni falsifié, ni copié tout ou partie de l'œuvre d'autrui, en particulier texte ou code informatique, dans le but de me 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 Nancy, le 14/01/2021

Signature :





## Déclaration sur l'honneur de non-plagiat

**Je, soussigné(e),**

**Nom, prénom :** Graindorge Bleunwenn

**Élève ingénieur(e) régulièrement inscrit en 2<sup>me</sup> année à TELECOM Nancy**

**Numéro de carte étudiante :** 31919591

**Année universitaire :** 2020/2021

**Auteur, en collaboration avec Chanteloup Marie-Astrid, Delaporte Lucille et Tilch Niels du rapport :**

### Projet Compilation - Langage Blood

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 Nancy, le 14 janvier 2021

Signature :



## Déclaration sur l'honneur de non-plagiat

**Je, soussigné(e),**

**Nom, prénom : TILCH Niels**

**Élève ingénieur(e) régulièrement inscrit en 2<sup>me</sup> année à TELECOM Nancy**

**Numéro de carte étudiante : 31916918**

**Année universitaire : 2020/2021**

**Auteur, en collaboration avec Chanteloup Marie-Astrid, Delaporte Lucille et Graindorge Bleunwenn du rapport :**

### Projet Compilation - Langage Blood

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 Nancy, le 14/01/2020

Signature :



# Chapitre 2

## Gestion de projet

### 2.1 Équipe de projet

L'équipe se compose de quatre étudiants en deuxième année :

- Marie-Astrid Chanteloup
- Lucille Delaporte
- Bleunwenn Graindorge
- Niels Tilch

Niels est désigné chef de projet, il aura la responsabilité d'animer les réunions et de suivre l'avancement du projet.

Marie-Astrid est désignée secrétaire de projet, elle aura la responsabilité de rédiger les comptes-rendus de réunions et de mettre à jour le rapport de projet sur la plateforme GitLab.

### 2.2 Analyse du projet

#### 2.2.1 Définition des objectifs

Les objectifs ont été définis à l'aide de la méthode SMART :

	Critère	Indicateur
S	Spécifique	L'objectif est défini clairement.
M	Mesurable	L'objectif est mesurable, par des indicateurs chiffrés ou livrables.
A	Atteignable	L'objectif doit être motivant sans être décourageant et doit apporter un plus par rapport au lancement du projet.
R	Réaliste	L'objectif doit être réaliste au regard des compétences et de l'investissement de l'équipe du projet.
T	Temporellement défini	L'objectif doit être inscrit dans le temps, avec une date de fin et des jalons.

#### 2.2.2 Analyse des risques : Matrice SWOT

Nous avons évalué les risques liés au déroulement du projet mais aussi les qualités de l'équipe et les opportunités liées au projet. Ils sont résumés dans la matrice SWOT présentées en figure 2.1



FIGURE 2.1 – Matrice SWOT

### 2.3 Organisation du projet

Le projet se découpe en deux phases. La première prend place du mois de septembre 2020 jusqu'au début du mois de janvier 2021. La deuxième prend place du mois de janvier 2021 au mois de avril 2021.

Pour la première partie du projet, nous avons découpé le projet en 24 étapes.

Elles ont été prévues dans le temps selon le diagramme de Gantt représenté en figure 2.2. Leur répartition entre les membres de l'équipe est présentée dans la matrice RACI en figure 2.3.

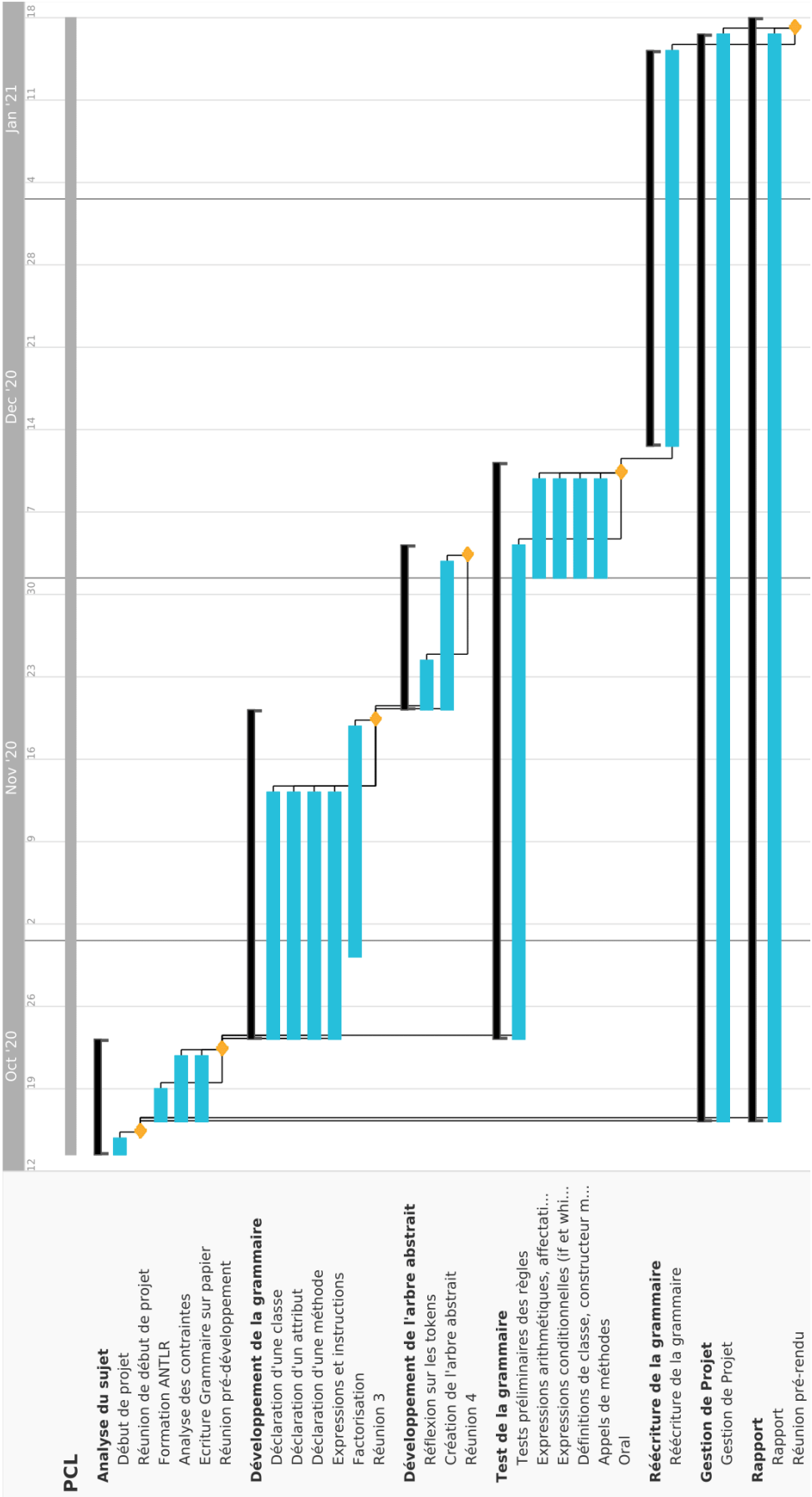


FIGURE 2.2 – Diagramme de Gantt

<div>Matrice RACI</div>									
		Niels	Marie-Astrid	Bleunwenn	Lucille				
Lots	Livrables ou taches PCL1								
Lot 1	Developpement								
Lot 1.1	Developpement de la grammaire								
Lot 1.1.1	Déclaration d'une classe	A	R	A	A				
Lot 1.1.2	Déclaration d'un attribut	A	A	A	R				
Lot 1.1.3	Déclaration d'une méthode	A	A	R	A				
Lot 1.1.4	Expressions et instructions	R	A	A	A				
Lot 1.2	Developpement de l'arbre abstrait								
Lot 1.2.1	Réflexion et mise en place des Tokens	R	A	R	A				
Lot 1.2.2	Ecriture de l'arbre abstrait	A	R	A	R				
Lot 1.3	Réécriture de la grammaire								
Lot 1.3.1	Réécriture de la grammaire	A	A	C	R				
Lot 2	Tests								
Lot 2.1	Tests de la grammaire et de l'AST								
Lot 2.1.1	Tests préliminaire des règles de grammaire	A	A	A	R				
Lot 2.1.2	Expressions arithmétiques, affectation	C	R	C	C				
Lot 2.1.3	Expression conditionnelles (boucles if et while)	C	C	R	C				
Lot 2.1.4	Définition de classe, constructeurs	R	C	C	C				
Lot 2.1.5	Appel de méthode	C	C	C	R				
Lot 3	Rapport								
Lot 3.1	GDP								
Lot 3.1.1	Compte-rendus de réunion	I	R	I	I				
Lot 3.1.2	Matrice RACI	I	I	R	I				
Lot 3.1.3	Diagramme de GANTT	R	I	I	I				
Lot 3.1.4	Charte de projet	I	I	I	R				
Lot 3.2	Rédaction et explication de la grammaire								
Lot 3.3	Rédaction et explication de l'AST								
Lot 3.4	Rédaction et explication des tests								
R	Réalisateur								
A	Approbateur								
C	Consultant								
I	Informé								

FIGURE 2.3 – Caption

## 2.4 Outils de travail

### 2.4.1 Réunion

L'équipe s'est réunie toutes les semaines de période scolaire le mercredi après-midi à des horaires variables, soit pour une réunion, soit pour session de travail collaboratif. Dans un premier temps, ces réunions ont eu lieu au sein du campus de TELECOM Nancy, puis à distance grâce à la plateforme Discord.

### 2.4.2 Partage du travail

Tout le long du projet, l'équipe a utilisé le répertoire GitLab fourni par l'école. Une branche `dev` a été créée pour l'écriture de la grammaire. Une branche `gdp` a été crée pour y consigner le rapport.

### 2.4.3 Rédaction du rapport

Le rapport a été rédigé sur ShareLatex pour permettre à tout les membres de compléter leurs éléments simultanément.

## 2.5 Comptes-rendu des réunions

### 2.5.1 Réunion du 14 octobre 2020

Présent	Absent	Heure	Lieu
Marie-Astrid Chanteloup Lucille Delaporte Bleunwenn Graidorge Niels Tilch	Personne	14h	TELECOM Nancy

#### Ordre du jour

- Répartition des rôles
- Prise en main de sujet :
  - Rendu
  - Langage et premier découpage de la grammaire
- Gestion de projet - Général

## Répartition des rôles

Niels est désigné chef de projet et créera le répertoire GitLab du projet.

Marie-Astrid est désignée secrétaire. Elle se chargera de créer le projet Overleaf et de rédiger les CR de réunions.

## Prise en main du sujet

### *Rendu*

L'équipe fait un point sur les différents rendus pour préparer le plan de travail.

Ces rendus sont résumés ci-dessous :

- 30/11/2020 : grammaire LL(1) et AST
- 15/01/2021 : Rendu du premier rapport de projet sur la création de la grammaire, l'AST, les tests réalisés et la GdP associée

### *Langage et premier découpage de la grammaire*

Blood est langage orienté objet, il est aussi prôné à de nombreuses récursivités gauches et possibilités de conflit lecture-écriture auxquelles il faudra faire attention.

L'équipe décide de découper la grammaire associée à Blood en 4 grandes sous-parties :

- Déclaration de classe (Marie-Astrid)
- Déclaration d'attribut (Lucille)
- Déclaration de méthode (Bleunwenn)
- Expressions et instructions (Niels)

Chacune de ces parties est attribuée à un membre de l'équipe qui devra commencer à imaginer comment cette grammaire pourra s'organiser.

En plus de ce découpage, chacun peut déjà réfléchir aux éléments importants pouvant nuire au caractère LL(1) de la grammaire.

## Gestion de projet - Général

### *Analyse des risques - SWOT*

L'équipe note les éléments de la SWOT (présentée en fig 2.1). Marie-Astrid est chargée de les mettre en page pour le rapport.

### *Temps de réunion et de travail*

En plus de travail personnel de chacun, l'équipe décide de se réunir tous les mercredis après-midis, en fonction des horaires de chacun. Ces temps seront soit des réunions, soit des temps de travail collectif pour régler les problèmes et blocages de chacun tout en avançant le travail. Cela permettra aussi à tout le monde d'avoir une meilleure vision du projet.

Il est aussi décidé quand cas de problème ou de travail en distanciel imposé, l'équipe utilisera la plateforme Discord pour se retrouver.

### *Objectifs et Gantt*

L'équipe se fixe pour objectif d'avoir une grammaire LL(1) environ 1 mois avant la soutenance, se donnant le temps de s'atteler à la préparation de l'AST lors du mois suivant. Niels sera chargé de créer un Gantt en fonction de ces objectifs.

### *RACI*

Bleunwenn est chargée de créer une RACI à partir des éléments du Gantt préparé par Niels.

## TO-DO LIST

- Création des zones de partage
  - GitLab : Niels
  - Overleaf : Marie-Astrid
- Travail sur la grammaire théorique :
  - déclaration de classe : Marie-Astrid
  - déclaration d'attribut : Lucille
  - déclaration de méthode : Bleunwenn
  - expression et instruction : Niels
- GdP et secrétariat
  - Mise en page de la matrice SWOT : Marie-Astrid
  - Préparation du Gantt : Niels
  - Préparation de la matrice RACI : Bleunwenn

*Prochaine réunion : 21/10/2020*

2.5.2 Réunion du 21 octobre 2020

Présent	Absent	Heure	Lieu
Marie-Astrid Chanteloup Lucille Delaporte Bleunwenn Graidorge Niels Tilch	Personne	14h	TELECOM Nancy

Ordre du jour

- 1. Avancement de la grammaire théorique
- 2. Lancement de la grammaire sur ANTLR
- 3. Rapport d'étonnement

Avancement de la grammaire théorique

Chacun fait un retour sur son avancement.

Déclaration de classe - Marie-Astrid

Le départ est assez simple à mettre en place grâce aux nombreux mots-clé. Cependant, la zone de bloc pose problème à cause de la nécessité de prendre en compte la présence ou non d'un constructeur. Lucille propose de travailler avec Marie-Astrid pour régler le problème après la réunion.

Déclaration d'attribut - Lucille

Cette partie est finalement assez courte et est quasi-finie. Lucille peut donc maintenant aider à la conception des autres parties de la grammaire.

Déclaration de méthode - Bleunwenn

La problématique de la différentiation entre méthode et constructeur apparaît très vite. Il va falloir factoriser pour éviter les conflits.

Expression et instruction - Niels

Il y a beaucoup de tâches à effectuer et à prendre en compte. Cela semble un peu dur à réaliser seul et Niels demande de l'aide. Lucille s'y attellera avec Niels. Niels propose de continuer la partie sur les expressions et instructions qu'il a déjà commencé et de laisser la partie sur la partie sur les envois de messages.

Lancement de la grammaire ANTLR

Les premiers TP de PCL ont commencé et Bleunwenn et Marie-Astrid ont déjà eu le leur. Elles ont commencé à travailler sur ANTLR pour travailler la grammaire. Certains éléments ne sont pas encore très clairs sur le fonctionnement d'ANTLR-Works, mais avec l'aide de gentils 3A et d'Internet, ça ira mieux.

Bleunwenn et Marie-Astrid ont commencé sur deux documents différents, mais pour permettre les factorisations et les liens entre les règles, il faut travailler sur un unique fichier, il sera donc créé et reprendra les éléments déjà fait pour les rassembler.

Pour tout le monde, il falloir mettre les éléments en ligne, commencer à vérifier la qualité LL(1) de la grammaire et vérifier qu'un code Blood est bien reconnu par la grammaire. Pour cela, Bleunwenn et Lucille se propose de reprendre l'exemple de code envoyé par Mme Collin pour le mettre à disposition du groupe. Elles effectueront la répartition du fichier en privé.

Rapport d'étonnement

Il a été décidé de rédiger un rapport d'étonnement au cours du mois de novembre. L'équipe se met d'accord pour les avoir rédigés pour la fin de la semaine 46.

TO-DO LIST

- Grammaire sur ANTLR
  - Tous : reprendre sa grammaire théorique et la mettre en place sur ANTLR
  - Marie-Astrid et Lucille : régler le problème constructeur dans fonctionnement du bloc de classe
  - Bleunwenn : réfléchir aux factorisation adaptées pour sa partie de grammaire
  - Niels et Lucille : reprendre la partie expression et instruction à deux pour mieux l'aborder
- Préparation des premiers éléments de test : Bleunwenn et Lucille
- Rédaction des rapport d'étonnement - Tous

Prochaine session de travail : 04/11/2020  
Prochaine réunion : 18/11/2020

2.5.3 Réunion du 18 novembre 2020

Présent	Absent	Heure	Lieu
Marie-Astrid Chanteloup Lucille Delaporte Bleunwenn Graidorge Niels Tilch	Personne	14h	Dicord

Ordre du jour

1. Grammaire
2. Travail sur l’AST
3. Test
4. Rédaction du rapport

Grammaire

La grammaire arrive à un stade quasi fini, il reste quelques petites erreurs à corriger pour permettre à l’ensemble du programme test de passer, mais le très gros du travail est fait.  
Pour la qualité LL(1) de la grammaire, de nombreux éléments ont été factorisés pour permettre d’éliminer des éléments de récursion gauche et de conflit lecture-écriture. La grammaire est LL(1).

Travail sur l’AST

Maintenant que la grammaire est prête, il est temps de commencer à travailler la transformation en AST.  
Pour cela un travail sur les tokens à faire apparaître sera à faire pour déterminer quels sont les éléments indispensables. Bleunwenn et Niels effectueront un session de travail à la suite de cette réunion pour faire ce travail  
Ensuite, la grammaire sera reprise pour ajouter les éléments de l’AST par Marie-Astrid et Lucille.

Test

Le fichier de test principal a bien été recopié par Lucille et Bleunwenn. A part quelques petits éléments, l’ensemble est bien reconnu par la grammaire.  
Il faut maintenant faire des fichiers de test plus courts mettant en valeur les différentes parties de la grammaire. L’équipe met en place un découpage en 4 types d’éléments qui sont répartis pour chacun des membres de l’équipe :

- Expression arithmétiques, affectations et déclarations de variables - Marie-Astrid
- Boucles conditionnelles - Bleunwenn
- Définition de classe, constructeur, méthodes et paramètres - Niels
- Appels de méthodes - Lucille

Rédaction du rapport

Des premiers éléments du rapport peuvent commencer à être remplis. Pour éviter d’être pris de court lors du rendu, il serait bon de commencer dès maintenant.

TO-DO LIST

- Grammaire : finir de vérifier les éléments du fichier général de test - Lucille
- Travail sur l’AST
  - Réflexion sur les tokens - Bleunwenn et Niels
  - Mise en place de l’AST - Lucille et Marie-Astrid
- Mise en place des tests - Tous (chacun sa partie)
- Rédaction - Tous

Prochaine session de travail : 21/11/2020  
Prochaine réunion : 02/12/2020

2.5.4 Réunion du 2 décembre 2020

Présent	Absent	Heure	Lieu
Marie-Astrid Chanteloup Lucille Delaporte Bleunwenn Graidorge Niels Tilch	Personne	14h	Discord



Ordre du jour

- 1. Grammaire
- 2. AST
- 3. Test

Grammaire

La grammaire est maintenant complète. Elle reconnaît l’entièreté du fichier de test proposé par Mme Collin.

Bleunwenn note que, en réunion avec Mme Collin, elle a soulevé l’éventualité que la grammaire soit trop factorisée et que cela nous nuise. Il est trop tard pour reprendre la grammaire dans de trop grandes mesures avec la semaine d’examens à venir, mais des éléments seront peut-être travaillé si les membres de l’équipe trouvent le temps.

AST

L’AST est quasi complet, il reste quelques éléments à reprendre vis-à-vis de l’ordre des noeuds de l’arbre final.

De même que pour la grammaire en général, la possibilité que la grammaire soit trop factorisée peut être un malus.

Test

Chacun a commencé à réaliser ses tests plus spécifiques, pour le moment aucun problème n’a été soulevé. Les limites connues sont bien présentes, mais aucunes nouvelles limites n’a été détectée.

TO-DO LIST

- AST à finir - Lucille et Marie-Astrid
- Test - Tous, à finir pour le 8/12 pour permettre des vérifications avant la soutenance
- Grammaire à retravailler si possible - Tous

*Soutenance le 09/12/2020*

2.5.5 Réunion du 9 décembre 2020

Présent	Absent	Heure	Lieu
Marie-Astrid Chanteloup Lucille Delaporte Bleunwenn Graidorge Niels Tilch	Personne	17h30	TELECOM Nancy

Ordre du jour

- 1. Retour soutenance
- 2. Reprise de la grammaire
- 3. Rédaction du rapport

Retour soutenance

La soutenance ne s’est pas aussi bien passée que prévue. La grammaire était en effet trop factorisée et a donc fourni un AST qui manquait de sémantique.

Il a été proposé au groupe de prendre une grammaire réalisée par un autre groupe pour palier au problème lors de la seconde partie du projet.

Reprise de la grammaire

Pour le moment, l’équipe envisage de reprendre la grammaire pour proposer quelque chose de fonctionnel de soi-même. Cependant, une deadline doit être fixée après laquelle nous accepterons la grammaire fournie par l’équipe enseignante. Conscients que les vacances de fin d’année ne sont pas des vacances reposantes ou particulièrement propices au travail, il est décidé de se donner deux semaines après la rentrée pour obtenir la nouvelle grammaire.

Lucille note qu’elle est très intéressée par le sujet et que travailler longuement dessus ne la gêne pas. Elle travaillera notamment lors de la deuxième semaine de vacances et ce sera l’occasion de la rejoindre si le reste de l’équipe le souhaite.

Rédaction du rapport

Bien que la grammaire soit à retravailler, les éléments introductifs peuvent être rédigés avec certains éléments de gestion de projet et des tests.  
Pour ce qui est des éléments qui concernent la grammaire, leur rédaction sera difficile sans sa possible nouvelle version.  
Un session de travail est prévue à la rentrée pour que chacun voit où se situer dans la rédaction.

TO-DO LIST

- Reprise de la grammaire - Tous
- Rédaction du rapport
  - Éléments introductifs, de gestion de projet et de test - Tous, dès que possible
  - Éléments en lien avec la grammaire - Tous, en attente

Prochaine réunion : 11/01/2021

2.5.6 Réunion du 11 janvier 2021

Présent	Absent	Heure	Lieu
Marie-Astrid Chanteloup Lucille Delaporte Bleunwenn Graidorge Niels Tilch	Personne	14h	Discord

Ordre du jour

1. Retour grammaire
2. Travail de rédaction
  - Fiche projet et introduction
  - Gestion de projet
  - Explication de la grammaire
  - AST
  - Test
  - Bilan

Retour grammaire

La grammaire a été retravaillée avec son arbre et est bien plus claire en termes de sémantique. C’est une bonne nouvelle!  
Lucille a rendez-vous avec M Oster dans la semaine qui suit pour vérifier que l’ensemble correspond aux attendus.

Rédaction

*Fiche projet et introduction*  
La fiche projet sera insérée dans le chapitre introductif du document.  
Le reste de l’introduction est déjà rédigé et il reste maintenant à chacun de remplir les éléments manquant dans les déclarations sur l’honneur de non-plagiat.

*Gestion de projet*  
Une majorité des éléments a été rédigée. Il manque encore quelques CR de réunion et des éléments visuels à réaliser (SWOT et RACI). Marie-Astrid finira de rédiger les CR de réunions et avec Bleunwenn s’occuperont des éléments visuels.

*Explication de la grammaire*  
Chacun des membres de l’équipe reprendra sa partie initiale pour l’expliquer.

*AST*  
La description de l’AST se fera à travers ses tokens : chacun sera remis en contexte et expliqué succinctement. Une fois ces explications faites, un classement sera réalisé pour permettre une meilleure organisation des explications.

*Test*  
Chacun expliquera le fonctionnement des tests qu’il a réalisé et les limites qui y sont associé vis-à-vis de la nouvelle grammaire.  
Il sera bon, notamment dans cette partie, de profiter de la commande `\verb` pour différencier les éléments de code des explications rédigées.

*Bilan*  
Le bilan de groupe est effectué et est consigné dans la table de la partie 6.2 de ce document.  
Chacun est chargé d’écrire son bilan personnel pour cette première partie de projet, ainsi que de remplir la table horaire de travail réalisé.

**TO-DO LIST - Rédaction de rapport**

- Introduction :
  - Insertion de la fiche projet - Bleunwenn
  - Remplissage des attestations de non-plagiat - Tous
- GdP :
  - Ajout de la SWOT et de la RACI - Bleunwenn et Marie-Astrid
  - Rédaction des CR - Marie-Astrid
- Grammaire : rédaction de sa partie - Tous
- Test : rédaction de sa partie (penser à vérifier le fonctionnement du test) - Tous
- Bilan :
  - Bilan personnel et remplissage de la table horaire - Tous
  - Bilan de groupe et éléments généraux - Marie-Astrid

*Prochaine réunion : 15/01/2020 - Non-consignée dans ce rapport*



## Chapitre 3

# Grammaire du Langage

### 3.1 Parties de la grammaire

Un programme en langage Blood est constitué de la manière suivante :

- Une liste, potentiellement vide, de définition de classe définie par la règle `classDef`.
- Un programme principal définie par la règle `. progPrincipal`.

La règle `progPrincipal` permet de vérifier qu'un bloc de classe est bien de la forme suivante : `bloc`, c'est-à-dire une liste pottentiellement vide d' `instructions` ou une liste non-vide de déclarations de variable suivie du mot-clé `is`, lui-même suivi d'une liste non-vide de `instructions`.

Le mot-clé `is` se trouve dans `instruction` et il faudra vérifier par la suite que l'utilisation du mot-clé `is` se fait bien selon la règle définie ci-dessus.

La grammaire du langage Blood est séparée, par la suite, en quatre parties :

- Déclaration d'une classe
- Déclaration d'un attribut
- Déclaration d'une méthode
- Expressions et instructions

### 3.2 Déclaration d'une classe

#### 3.2.1 La classe

Une déclaration de classe est lancée par la règle `classDef`. Celle-ci vérifie que la déclaration de classe respecte bien, dans l'ordre d'apparition, les conditions suivantes :

- commencer par le mot-clé `"class"`.
- être nommée selon les règles associées aux classes (commencer par une lettre majuscule).
- prendre en compte une liste de paramètres (qui peuvent formels ou non), par `formalOuPas` qui renvoi sur des `param` et `formalParam`, ou leur absence pas les marqueurs `"(" et ")"`.
- vérifier l'héritage ou non de la classe grâce à `choixHeritage`.

#### 3.2.2 Les paramètres

La règle `formalOuPas` vérifie le format des paramètres annoncés en choisissant l'une des deux règles suivantes :

- `formalParam`.
- `param`.

La règle `formalParam` vérifie le format des paramètres annoncés selon les règles suivantes dans cet ordre :

- présence du mot-clé `"var"`.
- un nom de variable respectant les règles pour les variables (commencer par une lettre minuscule).
- présence du marqueur clé `":"`.
- présence d'un nom de classe ou d'une constante grâce à la règle `choixNameClass`.

La règle `param` quant à elle, vérifie le format des paramètres annoncés selon les règles suivantes dans cet ordre :

- un nom de variable respectant les règles pour les variables (commencer par une lettre minuscule).
- présence du marqueur clé `":"`.
- présence d'un nom de classe ou d'une constante grâce à la règle `choixNameClass`.

#### 3.2.3 L'héritage

La règle `choixHeritage` permet de vérifier la possibilité d'héritage d'une super classe et de lancer le bloc de classe associé à sa déclaration. Cela se fait selon les règles ordonnées suivantes :

- si on trouve le mot-clé **"extends"**, on doit trouver ensuite un nom de classe, puis le mot-clé **"is"** puis un bloc de classe, appelé par la règle du même nom, entouré des marqueurs clés **"{"** et **"}"**.
- sinon, on trouve directement le mot-clé **"is"** et le bloc de classe, entouré des **"{ }"**, qui le suit.

### 3.2.4 Bloc de classe

La règle **blocDeClasse** permet de vérifier qu'un bloc de classe est bien d'une des formes suivantes :

- une déclaration d'attribut, expliquée dans la partie suivante (Partie 3.2), suivie d'un reste de bloc de classe, appelé par **blocDeClasse**.
- une déclaration de méthode, dénotée par le mot-clé **def**, qui appelle la règle **methodeOuCons** pour déterminer la suite.

### 3.2.5 Constructeur et méthodes

La règle **methodeOuCons** détermine si la méthode qui est en train d'être définie est un constructeur ou non, grâce aux règles **consDef** et **methodeDef** dont le fonctionnement est expliqué dans la partie 3.4, et la suite de code est adaptée en fonction du besoin de constructeur dans la suite du bloc grâce aux règles **blocSansCons**, si un constructeur vient d'être déclaré, et **blocDeClasse**, sinon. En effet, il faut un unique constructeur dans un bloc de classe.

Enfin, la règle **blocSansCons** permet d'enchaîner les déclarations d'attribut ou de méthode en fonction de la présence du mot-clé **"def"**. Les définitions d'attributs et de méthode peuvent être faite dans un ordre quelconque.

## 3.3 Déclaration d'un attribut

La déclaration d'un attribut se fait selon la règle **attribDef** et vérifie que la déclaration d'attribut est bien définie selon le modèle suivant :

- le mot-clé **var**.
- la possibilité d'avoir le mot-clé **static**.
- présence d'un nom d'attribut.
- le marqueur clé **":"**.
- présence d'un nom de classe ou d'une constante grâce à la règle **choixNameClass**.
- l'instanciation possible de l'attribut via la formule **":" expression**.

## 3.4 Déclaration d'une méthode

Une déclaration de méthode est lancée par la règle **methodeDef** pouvant être appelée, comme nous l'avons vu dans le paragraphe précédent, par la règle **methodeOuCons** ou **blocSansCons** si elle est précédée du mot-clé **def**. Cette règle vérifie que la déclaration de méthode respecte bien, dans l'ordre d'apparition les conditions suivantes :

- la présence facultative des mots-clés **"static"** et **"override"**.
- le nom de la méthode, commençant par une lettre minuscule (contrairement aux noms de classes).
- une liste de paramètres, définie comme une succession de **param** séparés par des virgules et encadrés par les marqueurs **"("** et **")"**. Ces marqueurs sont présents même si la liste des paramètres est vide.
- un appel au corps de la méthode grâce à la règle. **permClassNameAvecExpr**

La règle **permClassNameAvecExpr** permet de vérifier si la méthode a un type de retour et d'appeler son corps. Cela se fait selon les règles suivantes :

- si on trouve le marqueur clé **":"**, il sera suivi du type de retour de la méthode, correspondant à un nom de classe ou d'une constante **":" nomClasse**. Après le type de retour, on trouve un appel à la règle **devPerm** pour définir le corps de la méthode.
- Si la méthode ne renvoie pas de résultat, on trouve le mot-clé **"is"** suivi d'un appel à un **bloc** contenant le corps de la méthode.

La règle **devPerm** permet de définir le corps de la méthode. Elle vérifie que l'ordre suivant est respecté :

- si le corps de la méthode se réduit à une unique expression, le marqueur **":"** est suivi d'une **expression**, correspondant à la valeur retournée par la méthode.
- si le corps est complexe, le mot-clé **"is"** est suivi d'un **bloc** contenant le corps de la méthode .

La règle **bloc** permet de définir les blocs du langage qui sont définis de la même manière que le bloc jouant le rôle de programme principal, entourés des marqueurs **"{" "**"

## 3.5 Expressions et instructions

### 3.5.1 Règles sur les instructions

Les instructions sont lancées par la règle **bloc** et apparaissent de plus dans les règles **condition** et **whileBoucle**.

Le non-terminal **instruction** s'assure que le code vérifie une des conditions suivantes :

- La présence d'une **expression** à laquelle on peut affecter ou non une autre **expression**. Cette **expression** et cette potentielle affectation sont suivies du marqueur ";".
- La présence d'un "bloc" avec la règle **bloc**, comme défini ci-dessus.
- La présence d'un **return** suivi du marqueur ";"
- La présence d'une condition avec un 'if', un 'then' et un 'else' dans cette ordre avec la règle **condition**.
- La présence d'une boucle 'Tant que', composée d'un 'while' au début, d'une condition parenthésée et d'une instruction à la fin, est détecté avec la règle **whileBoucle**.

### 3.5.2 Règles sur les expressions

Les expressions sont séparées en plusieurs règles dû à une factorisation nécessaire. Ci-dessous est la liste des règles formant les expressions possibles du langage **Blood**.

Le non-terminal **expression** est le corps principal et permet de reconnaître une expression. Voici les règles suivantes (dans l'ordre) :

- Si l'expression est une **instanciation**, on vérifie que l'instanciation commence bien avec **new** suivi le nom de la classe puis les parenthèses (avec zéro ou plusieurs paramètres).
- Si l'expression est une **constante** avec le type **Integer** ou **String**.
- Si l'expression est une **condExpr**.

### 3.5.3 Les règles sur les boucles

La règle **condition**, appelée depuis **instruction**, vérifie si une condition est bien écrite de la manière suivante :

- Tout d'abord, le mot-clé **if**.
- Suivi d'une **condExpr**, définie ci-dessous.
- Suivi d'un mot-clé **then**.
- Puis d'une **instruction** comme définie ci-dessus.
- Finalement, suivie du mot-clé **else**.
- Et enfin, suivi d'une **instruction**.

Quant à elle, la règle **whileBoucle** vérifie qu'une boucle **while** du code **Blood** est définie et écrite de la manière suivante :

- Tout d'abord, le mot-clé **while**.
- Suivi d'une **condExpr** entourée des marqueurs clés "(" et ")".
- Finalement suivie du mot-clé **do**.
- Lui-même suivi d'un **bloc**.

Une **condExpr** est définie selon le modèle suivant :

- Une opération définie par **exprOpérateur**.
- Un potentiel marqueur de condition tel que : "=", "<>", "<", ">" suivi à nouveau d'un **exprOpérateur**.

### 3.5.4 Les règles sur les opérations et feuilles terminales

Les non-terminaux **exprOpérateur** et **expr2** permettent de détecter les calculs arithmétiques avec les symboles '/', '\*', '+' et '-'. Il faut noter que l'on a une priorité sur les opérations du faite que l'on a **exprOpérateur** qui appelle **expr2**.

Le non-terminal **exprFinale** permet de soit appeler un **atom** ou soit effectuer une sélection ou un envoi de message sur un **atom** :

- Si l'analyseur syntaxique détecte un envoi de message ou une sélection, alors on appelle en supplément de **atom**, **messOuSelect**.

Le non-terminal **messOuSelect**, est appelé par la règle **atom** et permet de détecter un envoi de message ou une sélection selon les modalités suivantes :

- Si **messOuSelect** est uniquement constitué du marqueur "." puis d'un nom de variable, cela signifie que l'on a affaire à une sélection.
- Si **messOuSelect** est constitué du marqueur "." puis d'un nom de variable suivi finalement d'un **envoie\_message**, on a alors affaire à un appel de méthode.
- Le fait que chacune des deux options précédentes soient suivie de **messOuSelect?**

Le non-terminal **envoie\_message** décrit la syntaxe d'un envoi de message, il est appelé par la règle **messOuSelect**. Cependant, cette règle ne décrit que le formalisme appliqué pour les paramètres d'un appel de méthode car le . **NAME** devant précéder les parenthèses contenant les paramètres se trouve dans **messOuSelect**. Cela permet de ne pas faire de redondances gauches dans la grammaire.

Le non-terminal **atom** permet à l'analyseur syntaxique, les expressions finales du programme. On a les conditions suivantes :

- S'il détecte un chiffre ou un nombre, **NB** est appelé.
- S'il détecte des guillemets, alors **STRING** est appelé, le token **STR** permet aussi une identification plus rapide du terminal **STRING**.
- S'il détecte un nom de variable, **NAME** est appelé et si cette variable est déclaré à l'aide de **' : '** suivi d'un nom de classe, alors **choixNameClass** est appelé après l'appel initial. Cette deuxième partie, **' : '** **choixNameClass**, correspond à une déclaration de variable sans le mot-clé **var**.
- S'il détecte une déclaration de variable avec **var** suivie du nom de la variable suivie de **' : '** et du nom de la classe, alors **formalParam** est appelé.
- Si un identificateur spécifique (**this**, **super** ou **result**) est détecté, alors le non-terminal **identificateur** est appelé.
- Si une constante (**String** ou **Integer**) est détectée, alors le non-terminal **constante** est appelé.
- Si un couple de parenthèses avec une expression ou un cast à l'intérieur, alors **factoParenthese** est appelé.
- Si une majuscule est détectée à la première lettre, alors le terminal **CLASSNAME** est appelé, pour définir les noms de classe.

### 3.6 Limites de la grammaire

Cette section est importante pour voir où sont les points trop permissifs de la grammaire. En effet, décrire ces points permettra de faciliter la suite du projet en analysant les points à améliorer et les blocages sémantiques qui seront nécessaires.

Tout d'abord, dans la définition d'un **bloc**. Un **bloc** peut être défini des manières suivantes :

- Soit par une liste d'**instructions** potentiellement vide entourée des marqueurs clés **'{'** et **'}'**
- Soit par une liste, non vide, de déclarations de variable (les déclarations de variables se trouvent dans **atom**) suivie du mot-clé **is** lui-même suivi d'une liste non vide d'**instructions**

Cette définition d'un **bloc** a été difficile à mettre en place car une **instruction** peut commencer par le terminal **NAME** et une déclaration de variable peut aussi commencer par ce même terminal. Pour résoudre ce problème, nous avons alors décidé de définir un **bloc** comme étant une liste, potentiellement vide, d'instructions entourées des marqueurs clés **'{'** et **'}'**. C'est la raison pour laquelle, le mot-clé **is** est défini dans les instructions. C'est aussi la raison pour laquelle la déclaration de variable est effectuée dans la règle **atom**, permettant ainsi de factoriser les **NAME** se trouvant au début d'une instruction et ceux se trouvant au début déclaration de variable.

Cette méthode de résolution soulève cependant un plusieurs problèmes :

- Il est possible d'avoir plusieurs **is** dans un même **bloc**
- Il est possible d'avoir une liste vide de déclaration de variable avant un **is**. Idem pour une liste vide d'**instruction** après un **is**
- Il est possible d'avoir des instructions avant un **is**
- Il est possible d'avoir des déclarations de variable après un **is**
- Les déclarations de variables peuvent être faites dans des expressions quelconques telles que des calculs (**condExpr** ou encore **exprOperateur**).

Par la suite, nous avons dû mettre en place les règles sur les opérations : **condExpr**, **exprOperateur** et **expr2**. Ces 3 règles permettent, comme décrit si dessus, de déterminer les expressions explicitant une condition ou un calcul (**'+'**, **'-'**, **'\*'** ...) mais ils permettent aussi de décrire les 'calculs' sur les **STRING** que sont ceux avec l'opérateur **'&'**.

En l'état de la grammaire, tous les opérateurs, que ce soit ceux concernant le calcul (**'+'**, **'/'** ...) ou celui concernant la concaténation de chaîne de caractères, peuvent s'appliquer sur tous les **atoms** décrits ci-dessus. Cela peut poser des problèmes sémantiques, car additionner des **STRING** n'a pas vraiment de sens.

De la même manière, les envois de message et les sélections peuvent s'appliquer sur n'importe quel **atom**. Cependant, effectuer une sélection sur un **NB**, un **STRING** n'a pas vraiment de sens même si ces actions sont actuellement autorisées par la grammaire.

Finalement, sur un **atom** peut s'appliquer, comme décrit ci-dessus, soit une sélection, soit un appel de méthode via la règle **messOuSelect**. Cette façon de faire entraîne le fait qu'en-dessous d'un **messOuSelect** se trouve alors dans le noeud le plus à gauche, l'atom sur lequel s'applique le message ou la sélection puis les autres correspondent aux message(s) et sélection(s) successive(s). Cette spécificité doit bien être prise en compte afin d'effectuer dans de bonnes conditions l'analyse sémantique.



## Chapitre 4

# Structure de l'arbre abstrait

Lors de l'arbre abstrait, nous avons créé plusieurs tokens dont voici les significations :

### 4.1 Base d'un programme

- **ROOT** : indique le début du programme.
- **BLOCPRINCIPAL** : correspond au bloc principal du programme.

### 4.2 Déclaration d'une classe

- **CLASSDEF** : correspond à l'initialisation d'une classe.
- **LISTFORMALPARAM** : liste de paramètres sous leur format formel.
- **EXTENDS** : note la présence d'une super-classe lors de la déclaration de classe.
- **BLOCCLASSE** : bloc interne de déclaration de classe, regroupe les déclarations d'attributs et de méthodes avec une unique déclaration de constructeur.

### 4.3 Déclaration d'un attribut et d'une méthode

- **ATTRIBDEF** : note la définition d'un attribut.
- **CONSDEF** : correspond à la déclaration d'un constructeur dans une classe. Il ne doit apparaître qu'une seule fois dans l'arborescence de déclaration d'une classe.
- **METHODEDEF** : note la définition d'une méthode, non constructeur.
- **LISTPARAM** : liste de paramètres sous le format minimal, c'est-à-dire juste une expression.
- **FORMALPARAM** : paramètres sous leur format formel (**var Classe parametre**).
- **PARAM** : indique la présence d'un paramètre défini par son nom et son type.
- **TYPERETOUR** : indique le type de retour d'une méthode.

### 4.4 Instruction et expression

- **BLOC** : correspond à l'introduction d'un bloc avec les accolades.
- **INSTRUCT** : correspond à une instruction.
- **DECLVARIABLE** : correspond à la déclaration d'une variable locale.
- **AFFECT** : correspond à une affectation.
- **PARANTHESE** : permet de regrouper les calculs au sein d'une même parenthèse afin d'assurer les priorités de calculs.
- **CAST** : correspond à l'introduction d'un cast avec le mot-clé 'as'.
- **INST** : correspond à une instantiation.
- **LISTPARAMAPPELCONSTR** : liste des paramètres utilisés dans l'appel d'un constructeur lors de l'instanciation d'une variable.
- **IF** : note la présence d'une boucle conditionnelle **if**.
- **THEN** : instruction à réaliser lors de la vérification de la clause d'un **if**.
- **ELSE** : instruction à réaliser lors de la non-vérification de la clause d'un **if**.
- **WHILE** : note la présence d'une boucle conditionnelle **while**.
- **MESSOUSELECT** : permet de regrouper un message ou une sélection et l'objet sur lequel il s'applique.
- **SELECT** : note la présence d'une sélection.
- **MESSAGE** : note la présence d'un appel de méthode.
- **LISTOPERATEURS** : liste des opérateurs utilisés dans les paramètres d'un appel de méthode.
- **STR** : indique la présence d'une chaîne de caractères.



# Chapitre 5

## Tests

Les tests ont été effectués sur la grammaire en fonction de quatre parties :

- Expressions arithmétiques, affectation et déclarations de variables.
- Boucles conditionnelles (if et while).
- Définitions de classe, constructeur méthodes et paramètres.
- Appels de méthodes.

### 5.1 Expressions arithmétiques, affectation et déclarations de variables

Pour cette partie, on utilise le fichier `test/test1` avec comme première règle `prog`.

Dans l'ordre d'apparition des éléments dans le fichier, nous parlerons de déclaration de variable, puis des tests sur les opérations arithmétiques et leur associativité. Ensuite nous parlerons du fonctionnement des expressions conditionnelles et enfin nous noterons quelques erreurs syntaxiques reconnues par la grammaire.

#### 5.1.1 Déclaration de variables

Les déclarations de variables doivent se faire avant le mot-clé `is`. Cependant elles sont actuellement acceptée par la grammaire et font partie des limites de la grammaire qui devront être réglées sémantiquement parlant. Nous vérifions que les différentes techniques de déclaration de variable sont bien reconnues par notre grammaire :

- la déclaration simple de forme `nomDeVar : nomDeClasse;` est reconnue
- la déclaration précédée du mot-clé `var`, de forme `nomDeVar : nomDeClasse;` est reconnue
- la déclaration suivie d'une instanciation par une expression est reconnue
- la déclaration suivie d'une instanciation par le mot-clé `new` et d'un nom de classe est reconnue

On note qu'une déclaration faite avec un marqueur clé différent de `:` n'est pas reconnue.

#### 5.1.2 Opérations arithmétiques

Cette partie concerne les opérations `+`, `-`, `*`, `/` et `%`. Nous devons vérifier qu'elle respecte bien l'ordre usuelle des opérations arithmétiques et les règles d'associativité.

Pour cela, on s'intéresse à l'opération suivante :  $1 * 2 + 3/4 - 5$ . La multiplication et la division seront effectuées en premier de manière séparée, suivie de l'addition et de la soustraction de manière associative.

On retrouve bien ce comportement décrit dans l'arbre produit par ANTLR suivant :

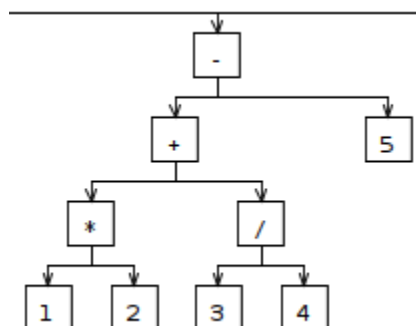


FIGURE 5.1 – Décomposition de l'opération  $1 * 2 + 3/4 - 5$  par la grammaire

De même pour l'association de chaînes de caractères grâce à l'opérateur `&`.

Le système de parenthèses décrit par une des possibilités de la règle `factoParentheses` permet de vérifier l'ordre de priorité classique des opérations : opérations entre parenthèses prioritaires sur toute opération extérieure aux parenthèses.

On note une limite au fonctionnement de la grammaire : il est possible de mélanger les opérateurs arithmétiques et ceux sur les chaînes de caractères ainsi que leur type associé dans une même opération.

5.1.3 Expressions conditionnelles

Pour travailler exclusivement sur le fonctionnement des expressions conditionnelles, on se place à l'intérieur de boucles conditionnelles "if" pour effectuer les tests.

On vérifie que chacun des signes d'opérations "<>", "<", ">" et "=" est bien reconnu grâce aux 4 boucles suivantes :

```
if a <> b then {} else {}
if 40 < 2 then {} else {};
if I > b then {} else {};
if pill = "pomme" then {} else {};
```

Ensuite on vérifie l'ordre de priorité dans les opérations conditionnelles avec la condition suivante : "alphabet + lila <> 12". Ici, l'élément d'inégalité doit être le dernier à être traité par rapport à l'opération arithmétique. Ce comportement est bien respecté comme le montre la décomposition arborescente de la boucle conditionnelle associée ci-dessous.

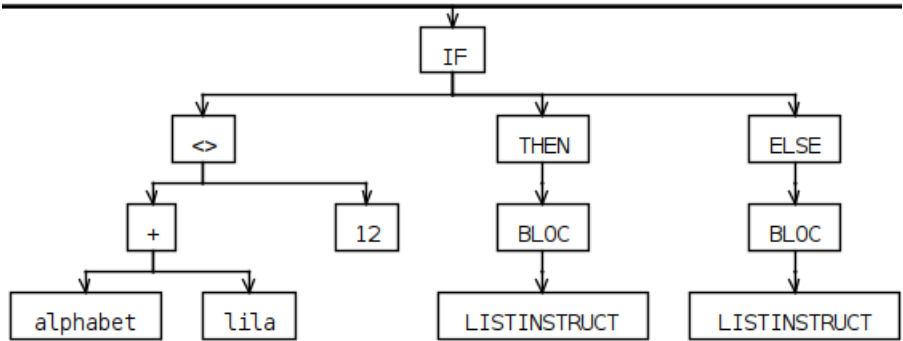


FIGURE 5.2 – Décomposition de la boucle conditionnelle présentée ci-dessus

5.1.4 Erreur de syntaxe soulevée par la grammaire

On note ici les erreurs de syntaxe que la grammaire détecte justement vis-à-vis des éléments étudiés dans cette partie.

- Une instruction composée d'expressions doit être terminée par le marqueur clé ";" pour être reconnue par la grammaire, sinon, la grammaire note une erreur

5.2 Boucles conditionnelles (if et while)

Cette partie de tests permet de vérifier la syntaxe des expressions conditionnelles, c'est à dire des boucles if et while. Ces tests se trouvent dans le fichier de ./tests/testBoucle.b. Nous parlerons dans un premier temps des boucles conditionnelles simples et de leur fonctionnement afin de nous concentrer sur les boucles imbriquées. Nous finirons par étudier quelques erreurs syntaxiques reconnues par la grammaire.

5.2.1 Boucles conditionnelles simples

On vérifie dans un premier temps les propriétés que le langage Blood doit autoriser et réussir à traiter :

- Une condition while avec un bloc vide

```
while (expression) do {}
```
- Une boucle if avec des blocs vides

```
if true then {} else {}
```
- Les boucles if contenant des instructions dans les blocs

```
if v = true then {
    "true".println();
}
else {
}
```
- Les boucles while contenant une succession d'instructions

```
while(n<2) do {
    NomClasse.methode();
    nomMethode.methode(a);
}
```

Comme nous l’avons vu, les blocs de nos conditions sont acceptés par la grammaire lorsqu’ils sont vides ou contiennent une succession d’instructions. Cela doit donc nous permettre d’imbriquer plusieurs boucles les unes dans les autres.

### 5.2.2 Les boucles imbriquées

Cette partie concerne la vérification du fonctionnement des boucles imbriquées. Pour vérifier que chaque imbrication fonctionne, nous avons testé que les conditions imbriquées étaient correctement appelées dans les blocs d’instructions des boucles appelantes. Voici quelques imbrications reconnues par la grammaire :

- Les boucles `else if`, revenant à l’utilisation de boucles `if` imbriquées :

```
if j = 12 then {
    while(n<2) do {}
}
else
    if true then{}
    else return;
```

- Les boucles `if` simples imbriquées

```
if n + 2 <> 20 then {
    if n>10 then {
        "a".println();
    }
    else {
        "b".println();
    }
}
else {}
```

- Des boucles `while` imbriquées dans des boucles `if`

```
if j = 12 then {
    while(n<2) do {}
}
else {}
```

- Des boucles `if` imbriquées dans des boucles `while`

```
while(i<10) do {
    if j = 12 then {}
    else {}
}
```

L’arbre abstrait de notre test fourni par ANTLR permet bien de confirmer la bonne imbrication des boucles :

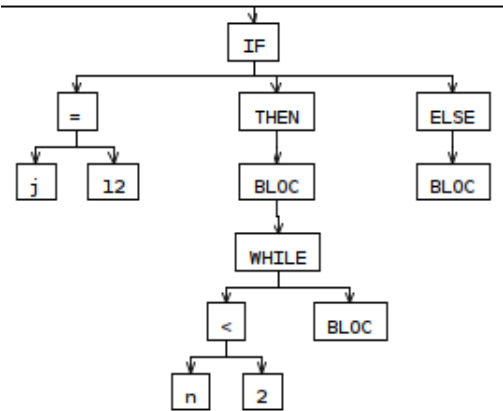


FIGURE 5.3 – Imbrication d’une boucle while dans une boucle if

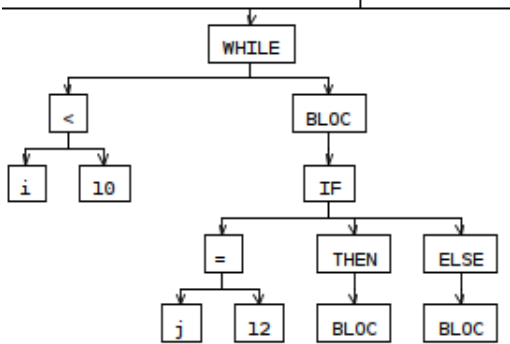


FIGURE 5.4 – Imbrication d’une boucle if dans une boucle while

### 5.2.3 Erreurs de syntaxe soulevées par la grammaire

Les tests nous permettent également de vérifier que la grammaire ne laisse pas passer certaines erreurs lors de l’écriture d’expressions conditionnelles :

- Une boucle `while` doit nécessairement se faire sur une condition, sans quoi elle ne peut pas être reconnue par la grammaire.  
`while () do {}`
- Une boucle `if` doit être suivie d’une condition `then` et d’une condition `else` pour être reconnue par la grammaire  
`if n=2 then {"ce test ne devrait pas passer".println();}`

### 5.3 Définitions de classe, constructeur méthodes et paramètres

Cette partie de tests va permettre de vérifier que la syntaxe est correctement respectée pour la déclaration de classe, la définition de méthode et leurs paramètres. Les explications suivantes concernent les propriétés que le langage Blood doit connaître.

Ici, on vérifie (dans le fichier de test `./tests/testClasse`) :

#### 5.3.1 Déclaration de classe

- La déclaration de classes avec plusieurs variables déclarées.  
`class Point(var x: Integer, var y: Integer, var name: String) is {}`
- La déclaration de classes avec plusieurs variables que l’on déclare.  
`class CouleurFactory() is {}`
- La déclaration de classes avec l’héritage "extends".  
`class PointColore(x: Integer, y: Integer, var_coul: Couleur) extends Points is {}`

#### 5.3.2 Déclaration de méthode

- La déclaration de méthodes avec (et uniquement avec) plusieurs variables et avec des instructions dans son bloc.  
`def Point(var x: Integer, var y: Integer, var name: String) is { this.next := this.next +1; }`
- La déclaration de méthodes n’ayant aucun paramètre et aucune instruction.  
`def print() is {}`
- La déclaration de méthodes ayant un type de retour (String ou Integer).  
`def egal(p: Point) : Integer is {}`

#### 5.3.3 Erreurs de syntaxe évitées par la grammaire

Il existe une multitude d’erreurs que la grammaire doit éviter dans cette partie :

- Des paramètres ou des variables déclarés sans noms.  
`class Bonjour(var :) is {}`
- Une classe suivie d’aucune parenthèses ni d’accolades ou d’un oubli du "is".  
`class Bonjour () is`  
`class Bonjour () {}`  
`class Bonjour {}`
- Une classe sans nom.  
`class () is {}`
- Un programme ne commençant pas par une classe.
- Une méthode dont la sortie présumée n’a pas de type.  
`def egal() : is {}`
- Une parenthèse ou accolade manquante.  
`def test( is {`
- L’héritage d’une classe n’ayant aucune super-classe.  
`class TestClasse () extends {}`

### 5.4 Appels de méthodes

Cette partie de tests va permettre de vérifier que la syntaxe est correctement respectée pour les appels de méthodes. Pour ce test, on utilise le fichier associé : `tests\testEnvoie`.

#### 5.4.1 Les appels de méthode successifs et leurs arguments

- Appel d’une méthode sur une variable.

- ```
point.envoi1();
```
- Envoi de message successifs sur une variable.  
`point.envoi1().envoi2().envoi3();`
  - Les envois de message sur tous les atoms existants.  
`point.carotte();`  
`GrandePomme.coin();`  
`"pomme".alphabet();`  
`1.jeTest();`  
`this.qqch();`  
`this.pomme.qqch();`  
`(1+2*6).methode();`  
`(as Class : variable).methode1();`
  - Les différentes possibilités pour les arguments d'un appel de méthode.  
`point.mal(unTruc);`  
`point.mal(unTruc, deuxTrucs);`  
`point.mal(unTruc.quiSapplique());`  
`point.mal(deuxTrucs.quiDiscute(), unPeu);`  
`point.mal(deuxTrucs.quiDiscute(), unPeu.quandMeme());`  
`point.desNombres(12, 14, "carotte", etDesPommes, Lili);`

### 5.4.2 Erreurs de syntaxe évitées par la grammaire

Il existe une multitude d'erreurs que la grammaire doit éviter dans cette partie :

- Les appels de méthodes doivent utiliser le marqueur '.'  
`point mal();`
- Les paramètres de méthode doivent être séparés par des virgules.  
`lila.carotte(jeMet pas deVirgules etCaMarchePas);`
- Une méthode doit être appelée sur un atom.  
`.carotte();`





# Chapitre 6

## Bilan du projet

### 6.1 Rapport d’étonnement 1 - Novembre 2020

Les rapports d’étonnement des membres ont été rédigés au cours des semaines 45 et 46 soit environ deux mois après le début du projet.

#### 6.1.1 Marie-Astrid CHANTELOUP

|                         |                                                                                                                                                                                                                |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Découverte du sujet     | - Le sujet est abordable, mais de nombreux éléments de récursion gauche sont problématiques.                                                                                                                   |
| Situation de travail    | - Les réunions en semi-présentiel n’ont pas toujours été simple à organiser. L’instauration du 2 <sup>me</sup> confinement aidera un peu l’organisation des réunions, mais vient avec son lot d’inconvénients. |
| Améliorations possibles | - Lorsque la situation s’améliorera, faire un choix entre présentiel et distanciel pour mieux aborder le travail.                                                                                              |

#### 6.1.2 Lucille DELAPORTE

|                         |                                                                                                                                                                                                                                              |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Découverte du sujet     | - Le sujet est adapté au cours que nous avons sur le sujet et les similitudes avec Java (propriétés des langages orientés objets) permettent une prise en main plus rapide mais qui peut parfois poser problème (différences très précises). |
| Situation de travail    | - Les réunions en semi-présentiels ont été plus difficiles à mettre en place que les réunions à distance.                                                                                                                                    |
| Améliorations possibles | - Définir des dates de réunions et lieux de réunions plus précis pour les réunions en semi-présentiels.                                                                                                                                      |

#### 6.1.3 Bleunwenn GRAINDORGE

|                         |                                                                                                                                                                                                    |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Découverte du sujet     | - Le langage présente des similitudes avec le Java, cela peut aider à s’y retrouver. La répartition du travail pour la création de la grammaire est facile à déterminer grâce aux quatres parties. |
| Situation de travail    | - La mise en place de binômes rend l’organisation de réunions et de séances de travail en commun en présentiel difficile.                                                                          |
| Améliorations possibles | - Mieux utiliser les outils pour travailler séparément en distanciel.                                                                                                                              |

#### 6.1.4 Niels TILCH

|                         |                                                                                                                                           |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| Découverte du sujet     | - Le sujet est très intéressant et mais de nombreux éléments commencent à nous bloquer comme la factorisation des règles de grammaire     |
| Situation de travail    | - La situation est difficile avec le confinement qui arrive, cela va être compliqué de s’adapter au distanciel avec le travail de groupe. |
| Améliorations possibles | - Avoir une communication accrue avec les membres du groupe est la priorité pour la bonne tenue du projet.                                |

6.2 Bilan de la première partie du projet

| Travail attendu                                                                                                                                                                              | Travail réalisé                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <div>- Écriture d’une grammaire pour le langage Blood</div> <div>- Obtention d’une grammaire LL(1)</div> <div>- Transformation vers un AST</div> <div>- Test de fonctionnement attendu</div> | <div>-Écriture d’une grammaire pour le langage Blood</div> <div>- Obtention d’une grammaire LL(1)</div> <div>- Transformation vers un AST</div> <div>- Test de fonctionnement attendu<ul style="list-style-type: none"><li>— Fonctionnement général</li><li>— Déclaration de classe</li><li>— Expression arithmétique et conditionnelles</li><li>— Déclaration d’attribut</li><li>— Déclaration de variables</li><li>— Utilisation des méthodes</li><li>— Boucles conditionnelles</li></ul></div> |
| <div>- Mise en place d’éléments de gestion de projet</div>                                                                                                                                   | <div>- Mise en place d’éléments de gestion de projet</div>                                                                                                                                                                                                                                                                                                                                                                                                                                        |

|                          | Points positifs                                                                                                                                                | Points négatifs                                                                                                                                                      | Expérience                                                                                                                                                                                                                          |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Gestion de projet</b> | <div>- répartition du travail adaptée au compétences de chacun</div> <div>- implication de l’ensemble du groupe au long terme</div>                            | <div>- travail en distanciel difficile</div> <div>- appui sur un membre volontaire du groupe qui a créé une légère perte de motivation pour le reste du groupe</div> | <div>- la répartition des tâches adaptée aux compétence a créé une efficacité sur l’avancement du projet</div> <div>- télé-travail difficile, mais donne de nouvelles clés pour mieux l’aborder dans le cadre de l’entreprise</div> |
| <b>Ecriture du code</b>  | <div>- organisation de mini-séance de travail quand un problème se faisait connaître</div> <div>- une forte entreaide très bien ressentie dans le groupe</div> | <div>- pas assez de recherche de contact et d’avis des professeurs qui a poussé à un mauvais premier rendu</div>                                                     | <div>- demander de l’aide en cas de blocage permet de trouver une solution plus rapidement</div> <div>- la recherche d’un regard extérieur permet de ne pas faire d’écart par rapport aux besoins</div>                             |

6.3 Bilan du projet par membre

6.3.1 Marie-Astrid CHANTELOUP

|                                |                                                                                                                                                                                     |
|--------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Points positifs</b>         | <div>- Équipe très complémentaire et agréable</div> <div>- Travail régulier malgré les conditions particulières</div>                                                               |
| <b>Difficultés rencontrées</b> | <div>- Problème de motivation personnelle</div> <div>- Manque de contact avec l’équipe enseignante</div> <div>- Rédaction du rapport tardive</div>                                  |
| <b>Expérience personnelle</b>  | <div>- Travail à distance</div> <div>- Travail sur le fonctionnement d’ANTLR et ANTLRWorks</div>                                                                                    |
| <b>Axes d’amélioration</b>     | <div>- Garder un contact plus régulier avec l’équipe enseignante/client pour éviter les effet tunnel</div> <div>- Commencer la rédaction plus tôt dans l’avancement du projet</div> |

6.3.2 Lucille DELAPORTE

|                                |                                                                                                                                                             |
|--------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Points positifs</b>         | <div>- Compétences et connaissances diversifiées</div> <div>- Investissement de la part de toute l’équipe et respect des deadlines</div>                    |
| <b>Difficultés rencontrées</b> | <div>- Demandes de vision extérieures trop peu nombreuses</div> <div>- Manques d’explications du code de ma part</div>                                      |
| <b>Expérience personnelle</b>  | <div>- Développement des compétences sur ANTLR</div> <div>- Développement des connaissances sur les grammaires et AST</div> <div>- Travail à distance</div> |
| <b>Axes d’amélioration</b>     | <div>- Développer les commentaires sur le code</div> <div>- Communiquer plus en détails sur les modifications effectuées</div>                              |

6.3.3 Bleunwenn GRAINDORGE

|                         |                                                                                                                                              |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| Points positifs         | - Équipe soudée et sérieuse, bonne ambiance de travail<br>- Disponibilité des membres du groupe                                              |
| Difficultés rencontrées | - Manque de motivation suite au confinement<br>- Contact avec les enseignants responsables trop tardif                                       |
| Expérience personnelle  | - Utilisation de ANTLRWorks<br>- Meilleure maîtrise des éléments du cours de Traduction des Lan-<br>gages                                    |
| Axes d'amélioration     | - Se renseigner plus régulièrement auprès des enseignants<br>- Se fixer plus d'objectifs à court terme pour ne pas perdre la motiva-<br>tion |

6.3.4 Niels TILCH

|                         |                                                                                                                          |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------|
| Points positifs         | - Une équipe compétente et agréable.<br>- Une communication active au sein du groupe.                                    |
| Difficultés rencontrées | - Une motivation qui baisse avec le distanciel et les mêmes problèmes rencontrés.                                        |
| Expérience personnelle  | - Communication avec des membres d'un groupe de projet en distan-<br>ciel.<br>- Meilleure connaissance d'un compilateur. |
| Axes d'amélioration     | -Essayer de remplir le rapport plus régulièrement.<br>- Se fixer sur un calendrier plus réaliste.                        |

6.4 Travail réalisé

Dans cette table est consigné l'estimation du temps en heures passé sur chacune des tâches du projet.

| Étapes                                                                          | Marie-Astrid | Lucille   | Bleunwenn | Niels     |
|---------------------------------------------------------------------------------|--------------|-----------|-----------|-----------|
| <b>Analyse du sujet</b>                                                         | <b>4</b>     | <b>4</b>  | <b>4</b>  | <b>4</b>  |
| Analyse du sujet                                                                | 2            | 2         | 2         | 2         |
| Préparation de la grammaire sur papier                                          | 2            | 2         | 2         | 2         |
| <b>Développement de la gram-<br/>maire</b>                                      | <b>8</b>     | <b>12</b> | <b>9</b>  | <b>16</b> |
| Déclaration d'une classe                                                        | 4            | -         | -         | -         |
| Déclaration d'un attribut                                                       |              | 4         | -         | -         |
| Déclaration d'une méthode                                                       | 1            | 2         | 4         | -         |
| Expressions et instructions                                                     |              | 2         | 2         | 8         |
| Factorisation                                                                   | 3            | 4         | 3         | 8         |
| <b>Développement de l'AST</b>                                                   | <b>5</b>     | <b>6</b>  | <b>6</b>  | <b>5</b>  |
| Réflexion sur les tokens                                                        | 1            | -         | 3         | 4         |
| Création de l'arbre abstrait                                                    | 4            | 6         | 3         | 1         |
| <b>Test sur la grammaire</b>                                                    | <b>5</b>     | <b>12</b> | <b>9</b>  | <b>6</b>  |
| Tests préliminaires                                                             |              | 6         | 6         | 4         |
| Expression arithmétiques, condi-<br>tionnelles et déclaration de va-<br>riables | 5            | 2         | -         | -         |
| Boucles conditionnelles                                                         |              | -         | 3         | -         |
| Définitions de classe                                                           |              | -         | -         | 2         |
| Appels de méthodes                                                              |              | 4         | -         | -         |
| <b>Réécriture de la grammaire</b>                                               | <b>2</b>     | <b>24</b> | <b>2</b>  | <b>2</b>  |
| Grammaire                                                                       | 1            | 14        | 1         | 2         |
| AST                                                                             | 1            | 10        | 1         | 0         |
| <b>Rédaction du rapport</b>                                                     | <b>18</b>    | <b>10</b> | <b>10</b> | <b>9</b>  |
| Partie introductive                                                             | 3            | 1         | 2         | 1         |
| Gestion de projet - Général                                                     | 3            | 2         | 2         | 3         |
| Gestion de projet - CR de réunion                                               | 4            | -         | -         | -         |
| Grammaire                                                                       | 2            | 3         | 2         | 2         |
| AST                                                                             | 1            | 1         | 1         | 1         |
| Test                                                                            | 2            | 2         | 2         | 2         |
| Bilan                                                                           | 3            | 1         | 1         | 1         |
| <b>Total</b>                                                                    | <b>42</b>    | <b>68</b> | <b>44</b> | <b>44</b> |