2019

Rapport du projet

ANALYSE DE DONEES STRUCTUREES

MICHAEL MARTIN – CHRISTIAN LIN

Table des matières

Partie 1

Afin de faire l'arbre de syntaxe abstraite, nous avons rigoureusement suivi les règles de la grammaire fournie. A savoir :

$$
\begin{aligned}
\textit{programme} &\rightarrow \textit{instruction}\ ;\ \textit{programme} \mid \varepsilon \\
\textit{instruction} &\rightarrow \texttt{Avancer}(\textit{expression}) \mid \texttt{Tourner}(\textit{expression}) \mid \texttt{Ecrire}(\textit{expression}) \\
\textit{expression} &\rightarrow \texttt{Lire} \mid \textit{nombre} \mid (\textit{expression}\ \textit{operation}\ \textit{expression}) \\
\textit{operation} &\rightarrow \texttt{+} \mid \texttt{-} \\
\textit{nombre} &\rightarrow \texttt{-?[1-9][0-9]*} \mid \texttt{0}
\end{aligned}
$$

Afin de la retranscrire le plus fidèlement possible, nous avons répartis les différents symboles terminaux en série de Sym (comme dans les précédents TPs) que nous avons transformé en Token grâce à notre Jflex.
Les symboles non terminaux constituent, quant à eux, une classe qui leur est propre. Nous avons donc un modèle comme suit :

La class **Program** qui possède un constructeur qui prend en argument une instruction et un programme. C'est dans cette classe que nous trouvons également une fonction run(Grid grid) qui permettra de faire tourner le programme plus tard.

La class **Instruction** qui est une classe abstraite. En effet, nous avons décidé d'utiliser cette méthode pour garder une certaine logique vis-à-vis de la grammaire. Dans cette même class nous trouvons la méthode exec(Grid grid) qui permettra à toutes les classes qui héritent d'Instruction d'avoir une méthode pour son utilisation. Nous avons donc les class **Avancer**, **Tourner**, et **Ecrire** qui héritent de la class Instruction. Ces dernières prennent donc un argument **Expression** et définissent dans chaque méthode exec, leurs façons de fonctionner.

La class **Expression** est structurellement similaire à la class Instruction puisque nous avons choisi de rester sur la même méthode et avoir un ensemble homogène. Nous avons donc la class **Lire** qui n'aura seulement qu'un exec avec sa fonction propre, la class **Nombre** qui s'occupe des valeurs ainsi que **Exp3** qui est la troisième option d'Instruction.

Pour finir, nous avons **Operation** qui est une class abstraite, comme Expression et Instruction afin de créer deux class **Plus** et **Moins** qui nous permettront d'être plus précis lors de la lecture des ordres.

Dans le parser (nommé **MyParser**) nous avons donc schématiser chaque choix de la grammaire par une nouvelle fonction qui vérifiera chaque Token grâce à notre reader **LookAhead1** venant d'un TP précédent. Il verifiera donc à chaque fois si le

Token est le bon et sera guidé à chaque Token. Si un problème survient, nous avons mis en place quelques Exceptions qui pourront indiquer les problèmes survenus.

Partie 2

Avec cette nouvelle partie, nous avons dû rajouter des choses à notre grammaire et donc à notre structure :

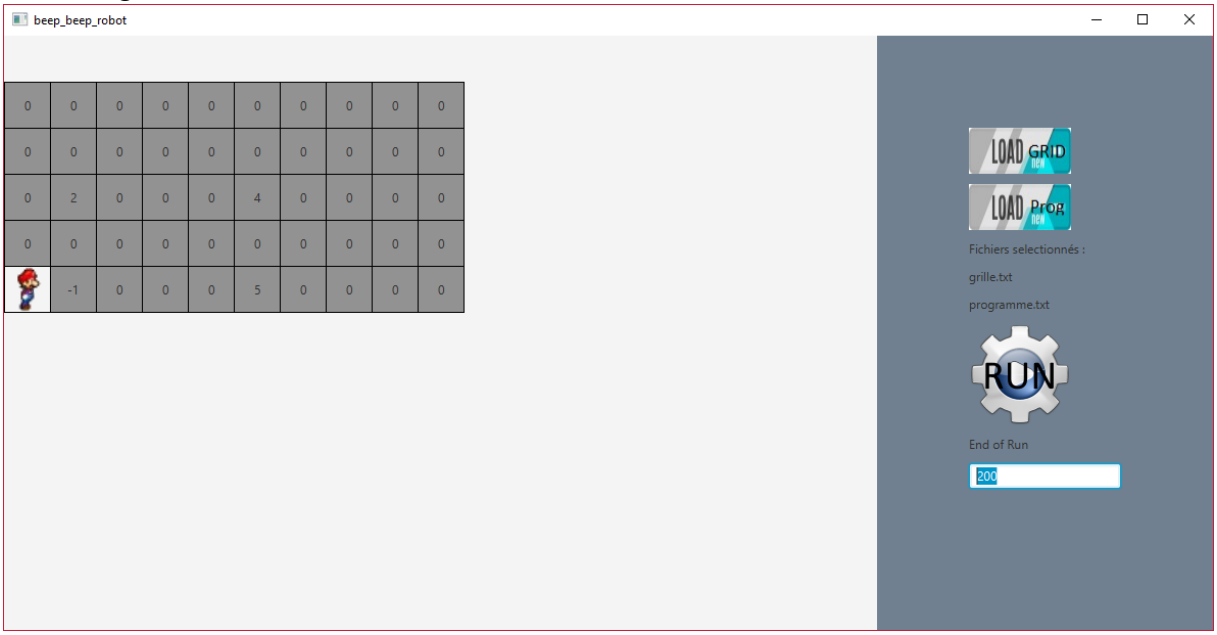
$$
\begin{aligned}
\textit{instruction} &\rightarrow \ldots \mid \texttt{Si}\ \textit{condition}\ \texttt{Alors}\ \textit{programme}\ \texttt{Fin} \\
&\quad \mid \texttt{TantQue}\ \textit{condition}\ \texttt{Faire}\ \textit{programme}\ \texttt{Fin} \\
\textit{condition} &\rightarrow \textit{expression}\ \textit{comparaison}\ \textit{expression} \\
&\quad \mid (\textit{condition}\ \textit{connecteur}\ \textit{condition}) \\
\textit{comparaison} &\rightarrow < \mid > \mid = \\
\textit{connecteur} &\rightarrow \texttt{Et} \mid \texttt{Ou}
\end{aligned}
$$

Nous avons donc suivi la méthode d'intégration que nous vous avons décrite dans la Partie 1 et nous avons rajouté quelques méthodes dans le Parser.

Nous avons cependant choisi (compte tenu du problème) de modifier les parenthèses dans condition afin d'en faire des accolades () => {} car ces dernières empêchaient la grammaire d'être LL1.

Partie 3

Dans cette partie, puisque nous avions le champ libre, nous avons choisi d'incorporer une interface graphique très simple, qui suivra les mouvements du robot dans la grille.



Ainsi qu'un caractère '#' qui représente un mur infranchissable pour le robot.