



KROP

Stuart Gueissaz
SI-MI4a
TPI



Table des matières

1	Analyse préliminaire	4
1.1	Introduction	4
1.2	Objectifs	4
1.3	Planification initiale	5
2	Analyse / Conception	6
2.1	Concept	6
2.1.1	Vue d'ensemble	6
2.1.2	MCD	8
2.2	Stratégie de test	10
2.3	Risques techniques	10
2.4	Planification	11
2.5	Dossier de conception	12
2.5.1	Maquettes / Use cases / Scénarios	12
2.5.2	MLD	18
2.5.3	Construction de l'arbre d'exécution	20
2.5.4	Gestion des pauses et arrêts d'un programme en cours d'exécution	22
2.5.5	Gestion du temps d'attente de la fin d'une animation	22
3	Réalisation	23
3.1	Dossier de réalisation	23
3.2	Description des tests effectués	24
3.2.1	Tests unitaires	24
3.2.2	Tests d'intégration	25
3.2.3	Tests d'acceptation par M. Carrel	26
3.3	Erreurs restantes	27
3.4	Liste des documents fournis	27
4	Conclusions	28
5	Annexes	29
5.1	Résumé du rapport du TPI / version succincte de la documentation	29
5.2	Glossaire	29
5.3	Historique des modifications	30
5.4	Sources – Bibliographie	31
5.5	Journal de bord	31
5.6	Manuel d'Installation	31
5.7	Manuel d'Utilisation	32
5.8	Archives du projet	34

1 Analyse préliminaire

1.1 Introduction

Pour mon TPI, j'ai comme mission de développer un outil utilisé lors des cours d'introduction à la programmation en première année du CPNV. Cette application se nommera Krop et elle devra remplacer l'outil actuel nommé PacmanProg. L'environnement de travail sera basé sur la refonte graphique de Krohonde que j'ai développée durant mon Pré-TPI. Krohonde a été développé en C# donc je continuerai à programmer avec le même langage

Le but de ce projet est de fournir un outil ayant les mêmes fonctionnalités que PacmanProg avec son propre langage de programmation sans devoir utiliser de Visual Basic. L'utilisateur pourra s'initier aux notions de base de la programmation telles que les branchements, les boucles, les sous-programmes et les variables en écrivant du pseudo code qui contrôle les actions et mouvements d'une fourmi.

1.2 Objectifs

L'objectif de base consiste à fournir les mêmes fonctionnalités que PacmanProg dans l'environnement Krohonde et avec un langage de programmation propre (plus de VB).

Plus précisément :

1. Définir la syntaxe d'un langage de programmation restreint couvrant :

- Les branchements conditionnels (if)
- Les boucles (for + while)
- Les variables
- La définition et l'utilisation de fonctions avec et sans paramètres
- L'évaluation d'expression

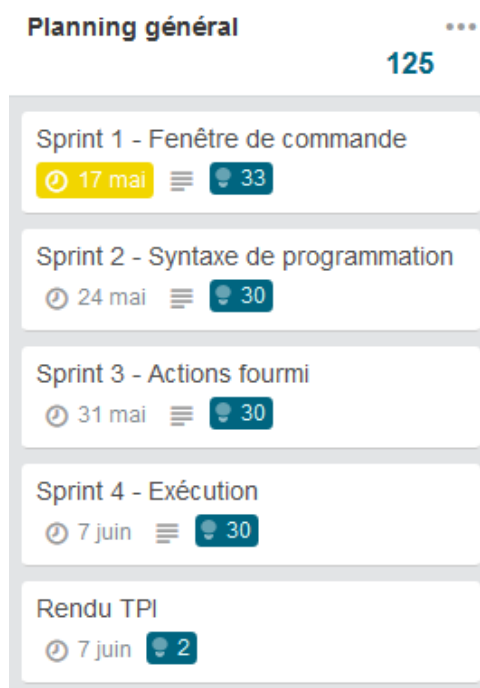
2. Établir un jeu de commande que la fourmi peut invoquer pour :

- S'immobiliser
- Se mettre en mouvement dans une direction donnée par un parmi 8 points cardinaux
- Détecter la proximité immédiate d'un obstacle
- Détecter une phéromone (elle doit être dessus)
- Poser une phéromone
- Supprimer une phéromone
- Émettre un message à l'utilisateur
- Demander à l'utilisateur d'introduire une valeur

3. Le programme lit, analyse et exécute un programme fourni sous forme de fichier texte

1.3 Planification initiale

Le TPI commence le mardi 8 mai 2018 à 8h50 et se terminera le jeudi 7 juin 2018 à 10h35. Ayant le méthode AGILE imposée par mon cahier des charges, j'ai décidé de découper ce projet en 4 sprints avec 2 périodes supplémentaire à la fin pour pouvoir faire une relecture de la documentation et faire la livraison du TPI. J'ai instauré un système de point dont un point équivaut à une période de 45 min.



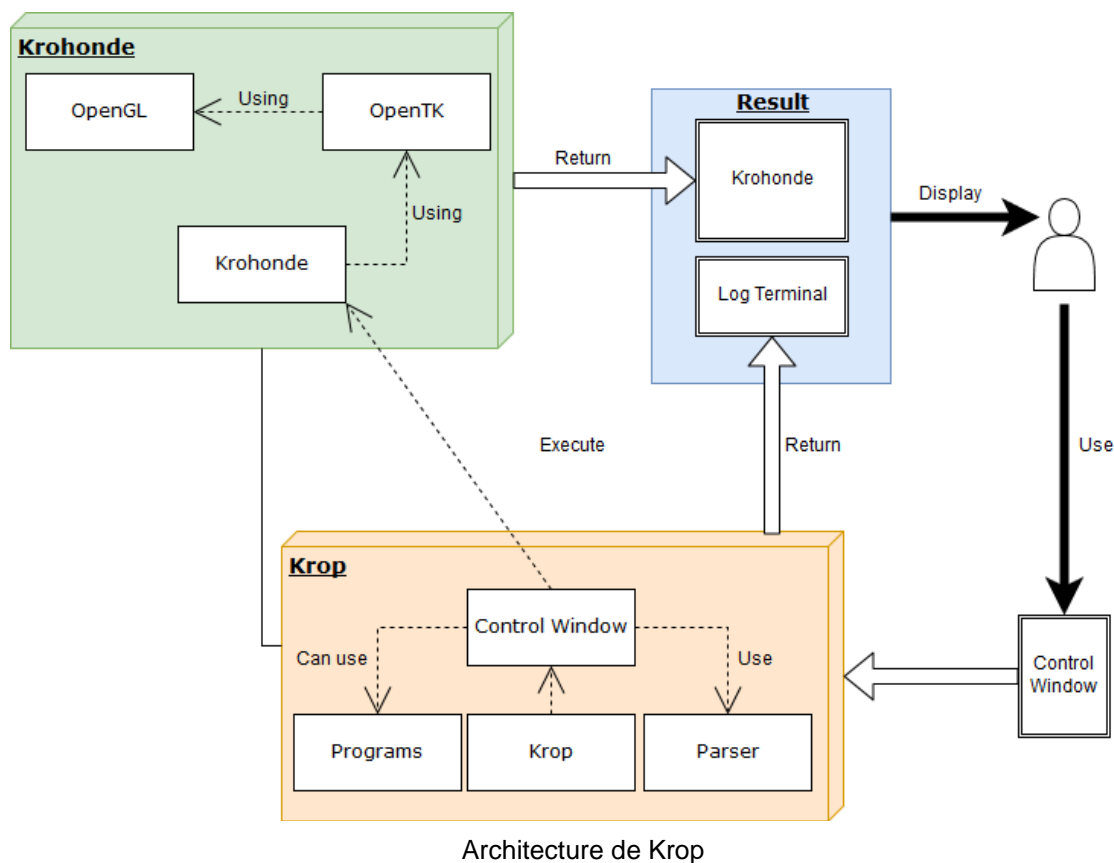
Plus de détails sont disponibles sur ce Trello : <https://trello.com/b/otGYQo2g/krop-tpi>

2 Analyse / Conception

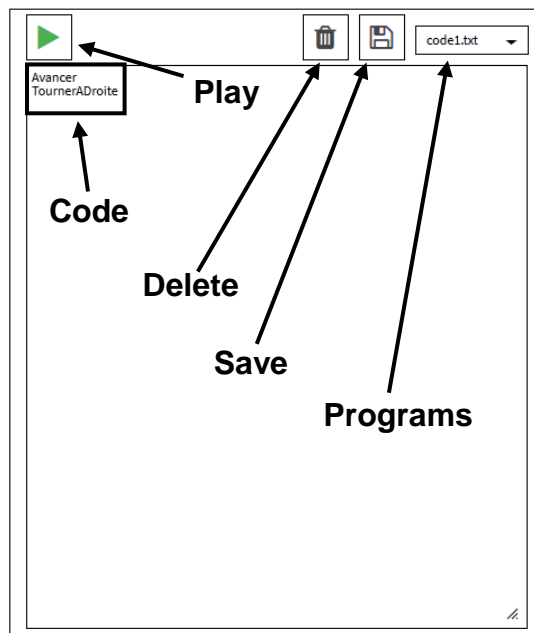
2.1 Concept

2.1.1 Vue d'ensemble

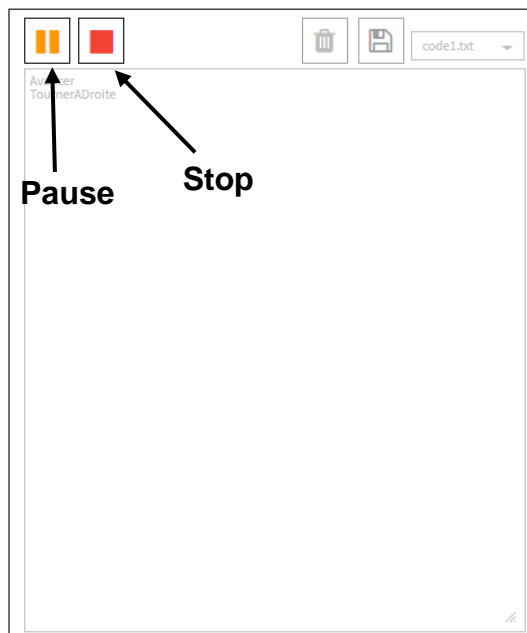
Pour ce projet, je vais devoir utiliser mon Pré-TPI Krohonde comme base. Krohonde utilise une librairie graphique OpenTK basée sur le moteur graphique OpenGL. Krop sera composé de trois fenêtres, la première sera la fenêtre de contrôle permettant de gérer les programmes et la deuxième sera la fenêtre de Krohonde affichant le résultat du programme en animant une fourmi dans un jardin et la dernière la console de logs. L'utilisateur interagira uniquement avec la fenêtre de contrôle et recevra les résultats en texte avec la console de logs et en graphique en voyant la fourmi bouger dans le jardin avec la fenêtre de Krohonde. Le programme de l'utilisateur sera d'abord analysé par le Parser pour vérifier qu'il n'y a pas d'erreurs syntaxiques et lexicales avant son exécution.



La fenêtre de contrôle permettra de créer, sauvegarder ou supprimer un programme en utilisant le bouton **Save** et **Delete**. Il sera aussi possible d'exécuter, mettre en pause ou stopper un programme en utilisant les bouton **Play**, **Pause** et **Stop**.

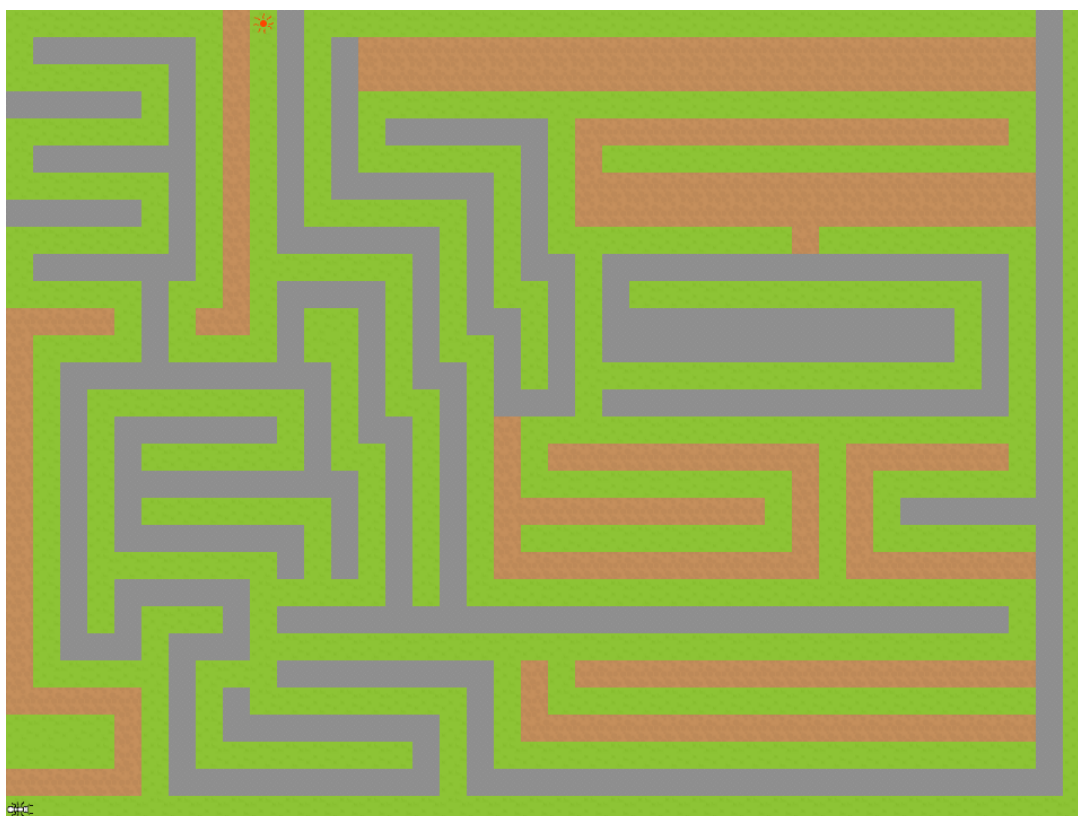


Fenêtre de contrôle avec un programme chargé



Fenêtre de contrôle avec un programme en exécution

La fenêtre de Krohonde affichera les résultats graphiquement en affichant les actions une fourmi dans le jardin en suivant les instructions du code exécuté comme par exemple à l'intérieur d'un labyrinthe.



La console de logs affichera les actions effectuées par l'application en format texte.

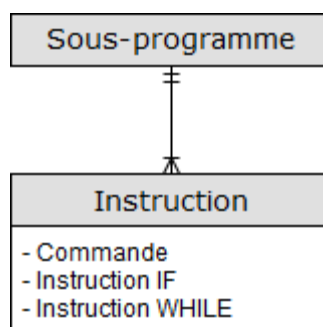
```

Loading control window
Resetting txtCode
Adding code1.txt to program list
Loading file code1.txt
Running program code1.txt
Code1
Ending program code1.txt
  
```

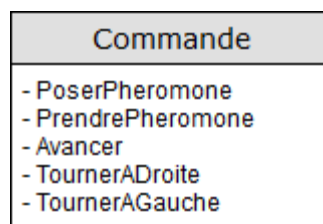
2.1.2 MCD

Krop, devant pouvoir exécuter un programme, doit avoir un arbre d'exécution généré à partir du code du programme. Pour se faire, il faut d'abord analyser la structure de ce code à l'aide d'un analyseur syntaxique et lexicale pour vérifier que la structure et les mots utilisés soient corrects. Le Parser pour fonctionner doit être régulé par un fichier de grammaire EBNF. Vous trouverez ci-dessous une explication schématisée de la grammaire de Krop.

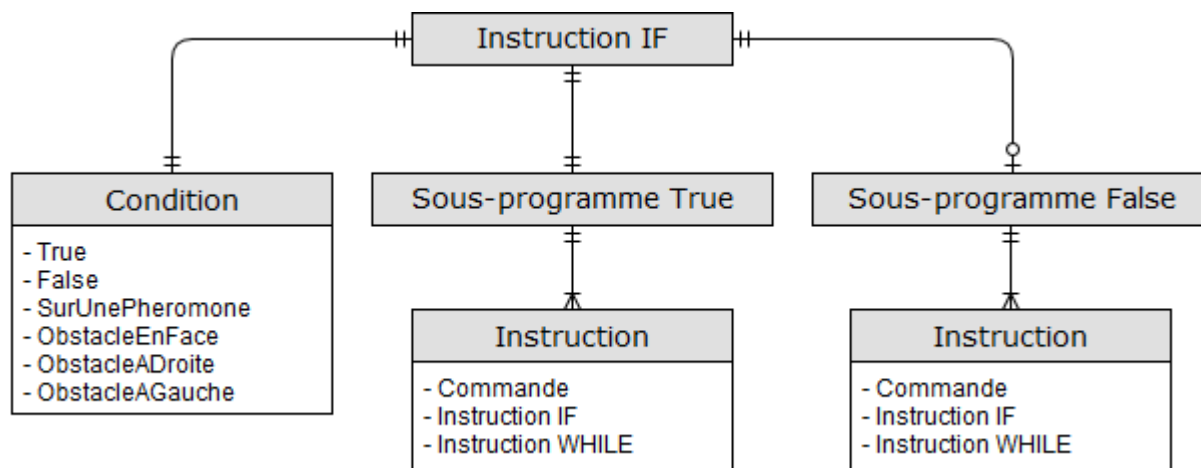
Un programme est composé de sous-programmes contenant des instructions. Les instructions peuvent être une commande, un branchement conditionnel (if) ou une boucle (while). Donc dans cette logique, la racine du programme est un sous-programme comprenant une ou plusieurs instructions.



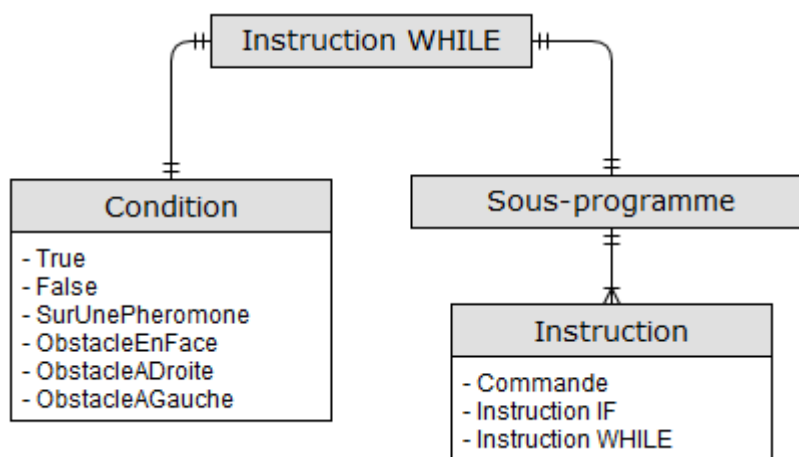
Une commande ne peut qu'être une des commandes définies pour Krop.



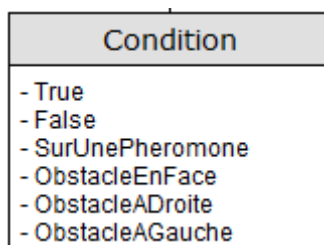
Un branchement conditionnel contient obligatoirement une condition et un sous-programme exécuté lorsque la condition est vrai. Le sous-programme, exécuté lorsque la condition est fausse, est facultatif.



Une boucle de type While contient une condition et un sous-programme.



Une condition peut être une condition définie pour Krop ou un booléen (True/False).



2.2 Stratégie de test

Pour ce TPI, j'ai décidé d'effectuer des tests unitaires, des tests d'intégration, et des tests d'acceptation avec M. Carrel. Les tests seront effectués sur mon ordinateur du CPNV.

À la fin du Sprint 1, je ferai :

- Des tests unitaires de la fenêtre de contrôle en suivant les scénarios de la fenêtre de contrôle
- Des tests d'acceptation lors du Sprint 1 review avec M. Carrel

À la fin du Sprint 2, je ferai :

- Des tests unitaires pour l'arbre d'exécution en essayant des fichiers de code qui feront office de scénario
- Des tests unitaires de la fenêtre de contrôle en suivant les scénarios de la fenêtre de contrôle
- Des tests d'acceptation lors du Sprint 2 review avec M. Carrel

À la fin du Sprint 3, je ferai :

- Des tests unitaires des commandes de la fourmi en testant chaque commande existante sans suivre de scénario
- Des tests d'acceptation lors du Sprint 3 review avec M. Carrel

À la fin du Sprint 4, je ferai :

- Des tests d'intégration de Krop composé de la fenêtre de contrôle, la fenêtre de Krohonde en essayant les scénarios de la fenêtre de contrôle et les fichiers de code utilisés lors du Sprint 2
- Des tests d'acceptation lors du Sprint 4 review avec M. Carrel

2.3 Risques techniques

Au moment où je rédige cette partie, j'ai déjà codé l'arbre d'exécution pour les commandes, les branchements conditionnel (if) et les boucles (while). Ce code peut déjà être utilisé avec la fenêtre de contrôle. Il me reste donc à faire l'implémentation de la fenêtre de Krohonde et à compléter l'arbre d'exécution avec les boucles (for), les variables, les fonctions et les expressions.

Je n'aurais peut-être pas le temps de tout faire car implémenter les variables, les fonctions et les expressions dans l'arbre d'exécution pourrait s'avérer être une tâche ardue. Je vais donc d'abord me concentrer sur les boucles (for) et l'implémentation de la fenêtre de Krohonde pour obtenir une version fonctionnelle de Krop puis je commencerai à implémenter les fonctionnalités manquantes.

2.4 Planification

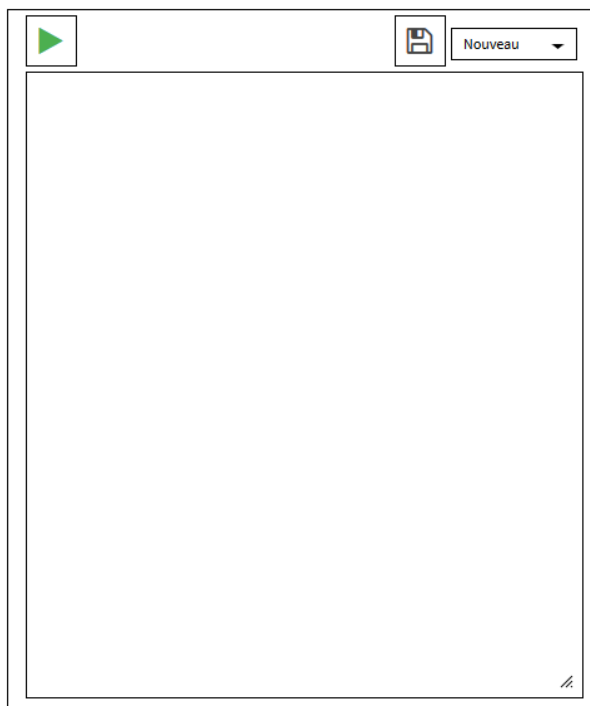
Ma planification se trouve sur ce Trello : <https://trello.com/b/otGYQo2q/krop-tpi>

	Date prévu	Date effective	Résultat
Sprint 1	17.05.2018	17.05.2018	Trello bien utilisé (gestion des points) Doc : inclure les scénarios + adapter aux changements d'UI Les scénarios de manipulation de scripts sont OK L'exécution simple est OK La pause/reprise n'est pas évaluable pour le moment
Sprint 2	24.05.2018	25.05.2018	Fonctions de pause/reprise/interruption OK Les éléments suivants fonctionnent: - commandes simples - evaluation de booleen (constantes, fonction et opérateur NOT, mais pas encore de comparaison) - boucle while - branchement if (avec le else) Bonne série de scénarios de test
Sprint 3	31.05.2018	31.05.2018	Revue effectuée en avance hier. Tout fonctionne bien et la documentation est à jour
Sprint 4	07.06.2018	07.06.2018	Sprint réussi

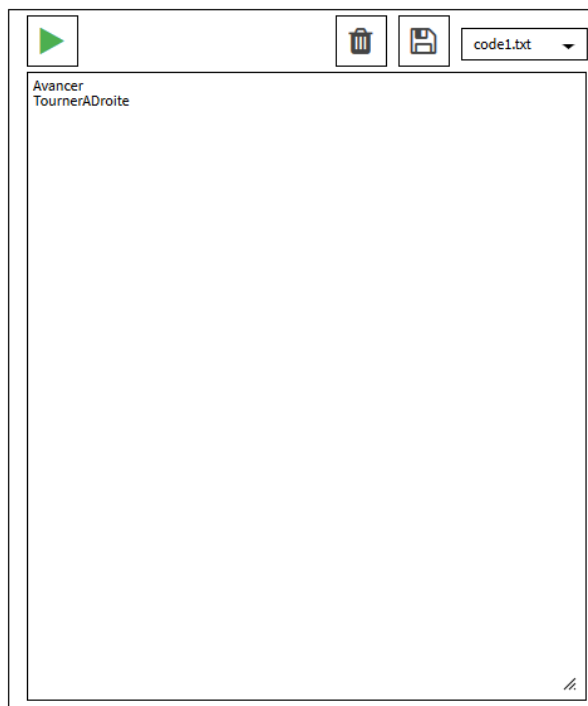
2.5 Dossier de conception

2.5.1 Maquettes / Use cases / Scénarios

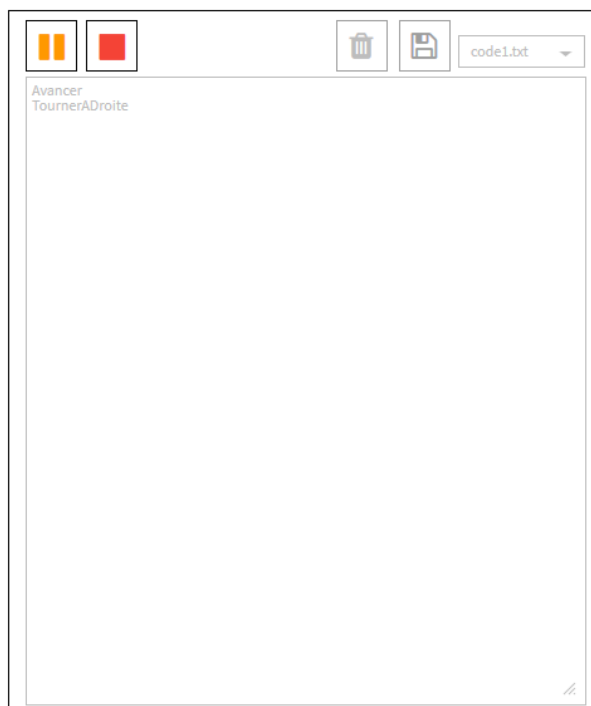
2.5.1.1 Fenêtre de contrôle



Fenêtre de contrôle par défaut



Fenêtre de contrôle avec un programme chargé



Fenêtre de contrôle avec un programme en exécution

Exécuter un programme sans fichier	
Action	Réaction
Exécute krop.exe	Affiche le jardin et la fenêtre de contrôle par défaut
Écrit Avancer dans la textBox	
Clique sur le bouton Play	<ul style="list-style-type: none"> ➤ Cache le bouton Play ➤ Affiche le bouton Pause et Stop ➤ Bloque et grise la comboBox, la textBox, les boutons Save et Delete ➤ Écrit chaque action dans la console de Krop

Charger un programme	
Action	Réaction
Exécute krop.exe	Affiche le jardin et la fenêtre de contrôle par défaut
Sélectionne code1.txt dans la comboBox	<ul style="list-style-type: none"> ➤ Affiche le programme dans la textBox ➤ Affiche le bouton Delete

Exécuter un programme	
Action	Réaction
Effectue le scénario Charger un programme	Affiche le jardin et la fenêtre de contrôle avec le programme code1.txt chargé
Clique sur le bouton Play	<ul style="list-style-type: none"> ➤ Cache le bouton Play ➤ Affiche le bouton Pause et Stop ➤ Bloque et grise la comboBox, la textBox, les boutons Save et Delete ➤ Écrit chaque action dans la console de Krop

Créer un programme	
Action	Réaction
Exécute krop.exe	Affiche le jardin et la fenêtre de contrôle par défaut
Écrit Avancer dans la textBox	
Clique sur le bouton Save	Affiche le formulaire Nouveau Programme
Écrit code2	
Clique sur le bouton Valider	Sauvegarder le programme code2 dans le fichier code2.txt dans le répertoire Code à la racine de krop.exe

Sauvegarder un programme	
Action	Réaction
Effectue le scénario Charger un programme	Affiche le jardin et la fenêtre de contrôle avec le programme code1.txt chargé
Ajoute une nouvelle ligne au programme en écrivant Stop	
Clique sur le bouton Save	Sauvegarder le programme code1.txt dans le fichier code1.txt dans le répertoire Code à la racine de krop.exe

Supprimer un programme	
Action	Réaction
Effectue le scénario Charger un programme	Affiche le jardin et la fenêtre de contrôle avec le programme code1.txt chargé
Clique sur le bouton Delete	Affiche un pop-up de confirmation
Clique sur le bouton Valider	<ul style="list-style-type: none"> ➤ Supprime le fichier Code/code1.txt ➤ Supprime code1.txt de la comboBox ➤ Affiche la fenêtre de contrôle par défaut

Stopper un programme	
Action	Réaction
Effectue le scénario Charger un programme	Affiche le jardin et la fenêtre de contrôle avec le programme code1.txt chargé
Effectue le scénario Exécuter un programme	Lance le programme code1.txt
Clique sur le bouton Stop	Stop l'exécution du programme et reviens à la situation après l'étape 1

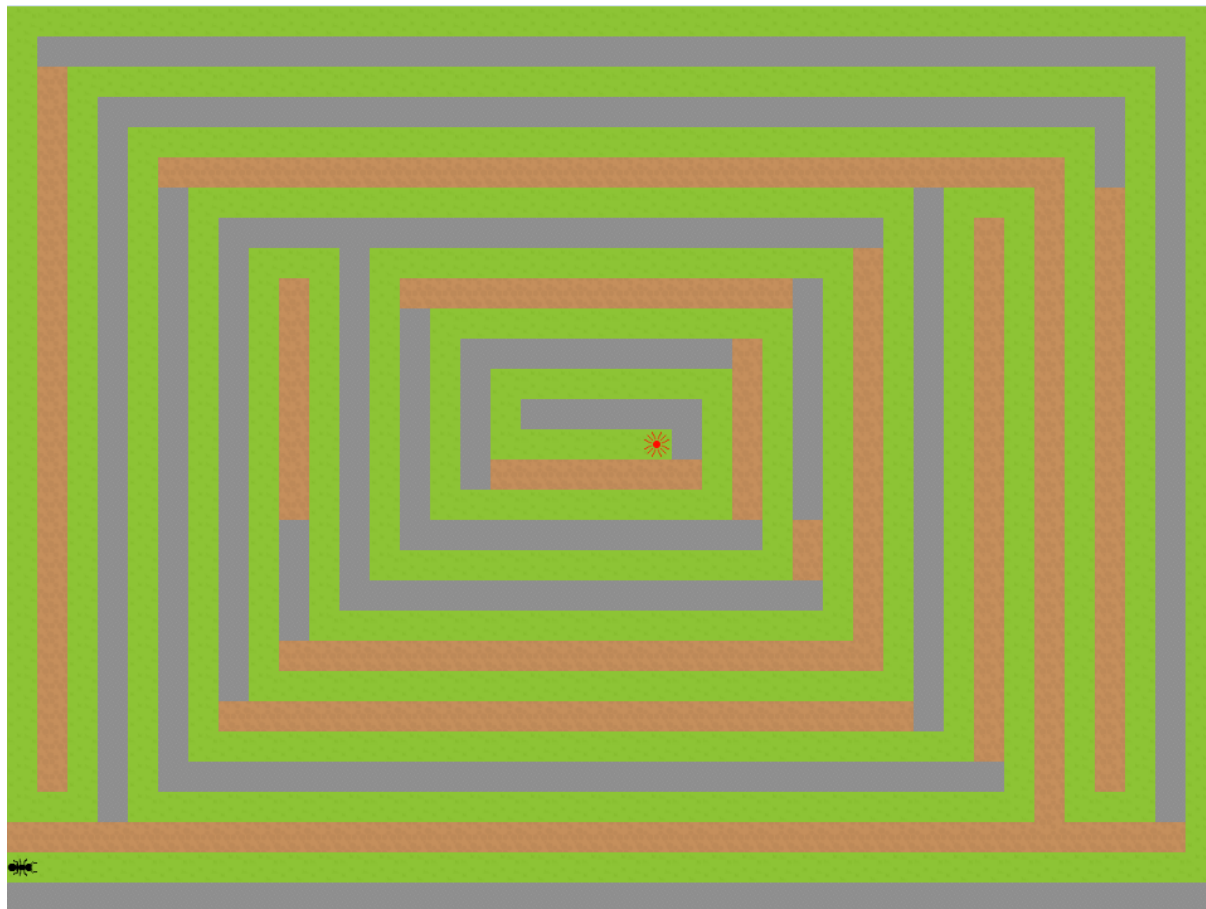
Mettre en pause et reprendre un programme	
Action	Réaction
Effectue le scénario Charger un programme	Affiche le jardin et la fenêtre de contrôle avec le programme code1.txt chargé
Effectue le scénario Exécuter un programme	Lance le programme code1.txt
Clique sur le bouton Pause	<ul style="list-style-type: none"> ➤ Met en pause l'exécution du programme ➤ Remplace le bouton Pause par le bouton Play
Clique sur le bouton Play	<ul style="list-style-type: none"> ➤ Reprend l'exécution où il s'est arrêté ➤ Remplace le bouton Play par le bouton Pause

2.5.1.2 Arbre d'exécution

	Codes Faux		
	Action	Condition particulière	Réaction
CodeFaux01	avancer	Il manque un ;	Message d'erreur
CodeFaux02	avancer; tourneradroite; tourneragauche; poserpheromone; prndrepheromone;	Prndrepheromone a une faute lexical	Message d'erreur
CodeFaux03	if(true){ avancer; }	Il manque une)	Message d'erreur
CodeFaux04	if(false){ avancer; }	Il manque un }	Message d'erreur
CodeFaux05	if(not vrai){ avancer; }	Le mot vrai n'est pas une condition	Message d'erreur

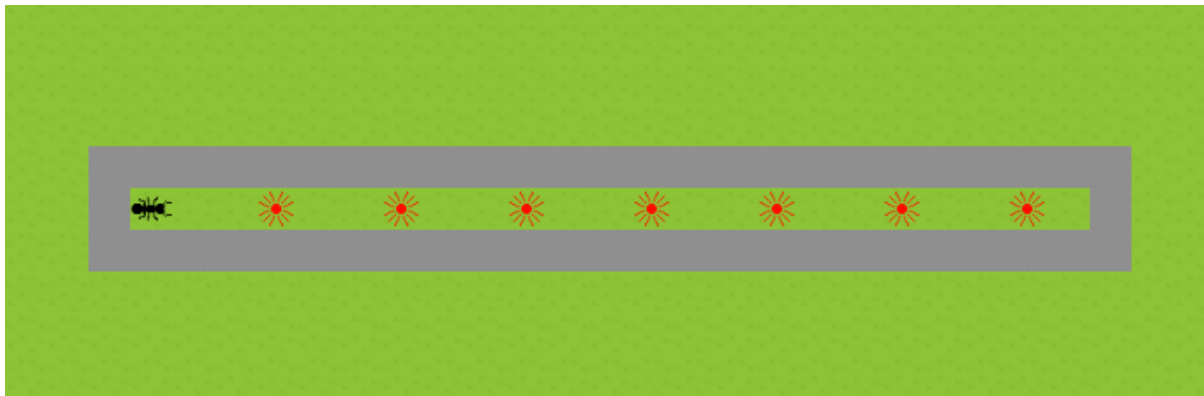
	Codes Justes	
	Action	Réaction
CodeJuste01	avancer;	Affiche : avancer
CodeJuste02	avancer; tourneradroite; tourneragauche; poserpheromone; prendrepheromone;	Affiche : avancer tourneradroite tourneragauche poserpheromone prendrepheromone
CodeJuste03	if(true){ avancer; }	Affiche : avancer
CodeJuste04	if(false){ avancer; }	N'affiche rien
CodeJuste05	if(not true){ avancer; }	N'affiche rien
CodeJuste06	if(not false){ avancer; }	Affiche : avancer
CodeJuste07	if(true){ avancer; } else{ tourneradroite; }	Affiche : avancer
CodeJuste08	if(false){ avancer; } else{ tourneradroite; }	Affiche : tourneradroite
CodeJuste09	if(true){ if(true){ avancer; } } else{ tourneradroite; }	Affiche : avancer
CodeJuste10	while(true){ avancer; }	Ne fait que d'afficher : avancer
CodeJuste11	while(false){ avancer; } tourneradroite;	Affiche : tourneradroite
CodeJuste12	while (true) { while(not obstacleenface) { avancer; } tourneragauche; }	Affiche 1 fois sur 10 avancer sinon ne fait que d'afficher tourneragauche

2.5.1.3 Le labyrinthe couloir



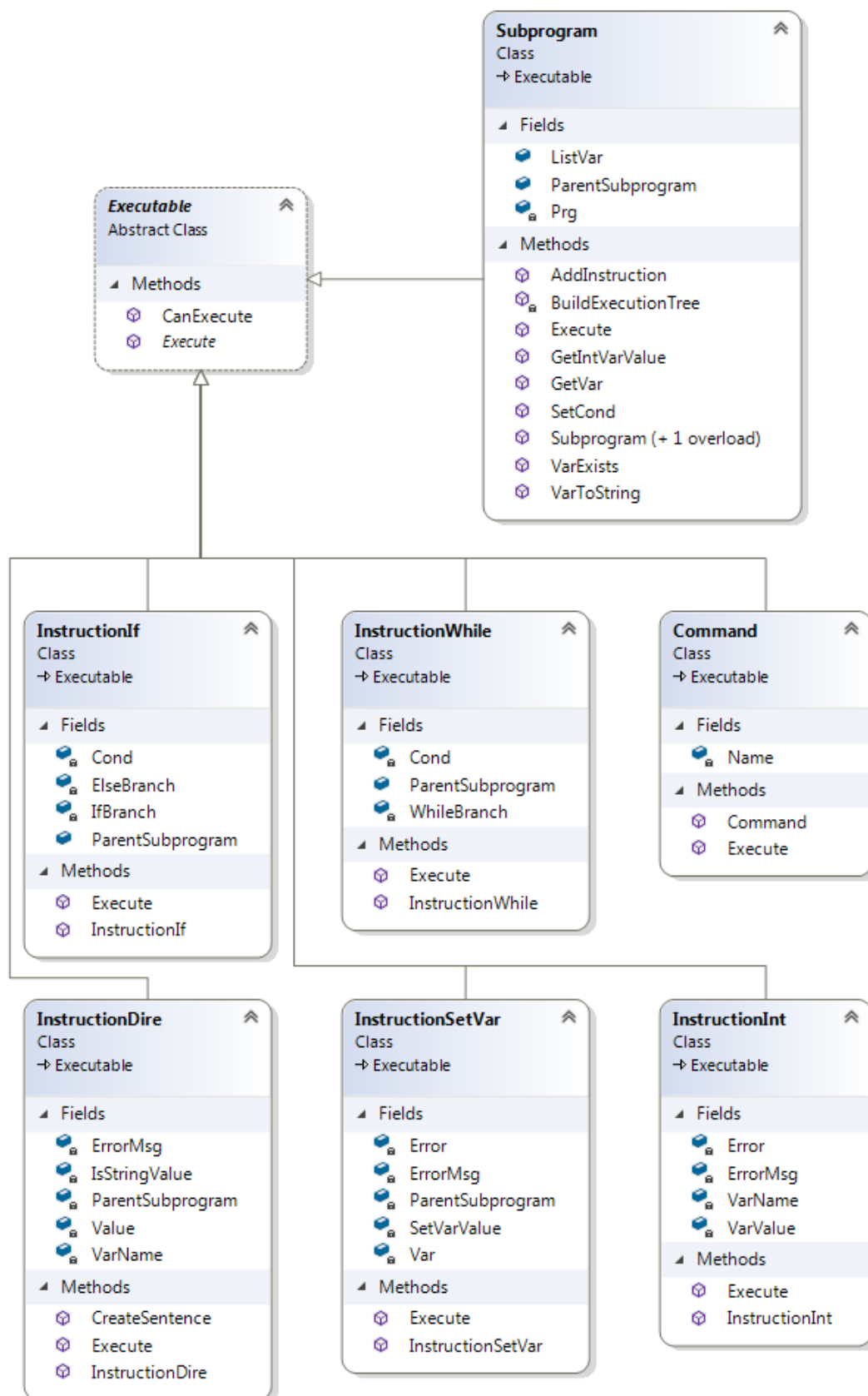
Labyrinthe Couloir	
Code	Réaction
<pre> while(true){ while(not obstacleenface){ if(not surunepheromone) { poserpheromone; } else { prendrepheromone; } avancer; } if(obstacleadroite){ tourneragauche; } else{ tourneradroite; } } </pre>	<p>La fourmi résous le labyrinthe en boucle tout en posant une phéromone sur sa case s'il n'y en a pas sinon elle prend la phéromone</p>

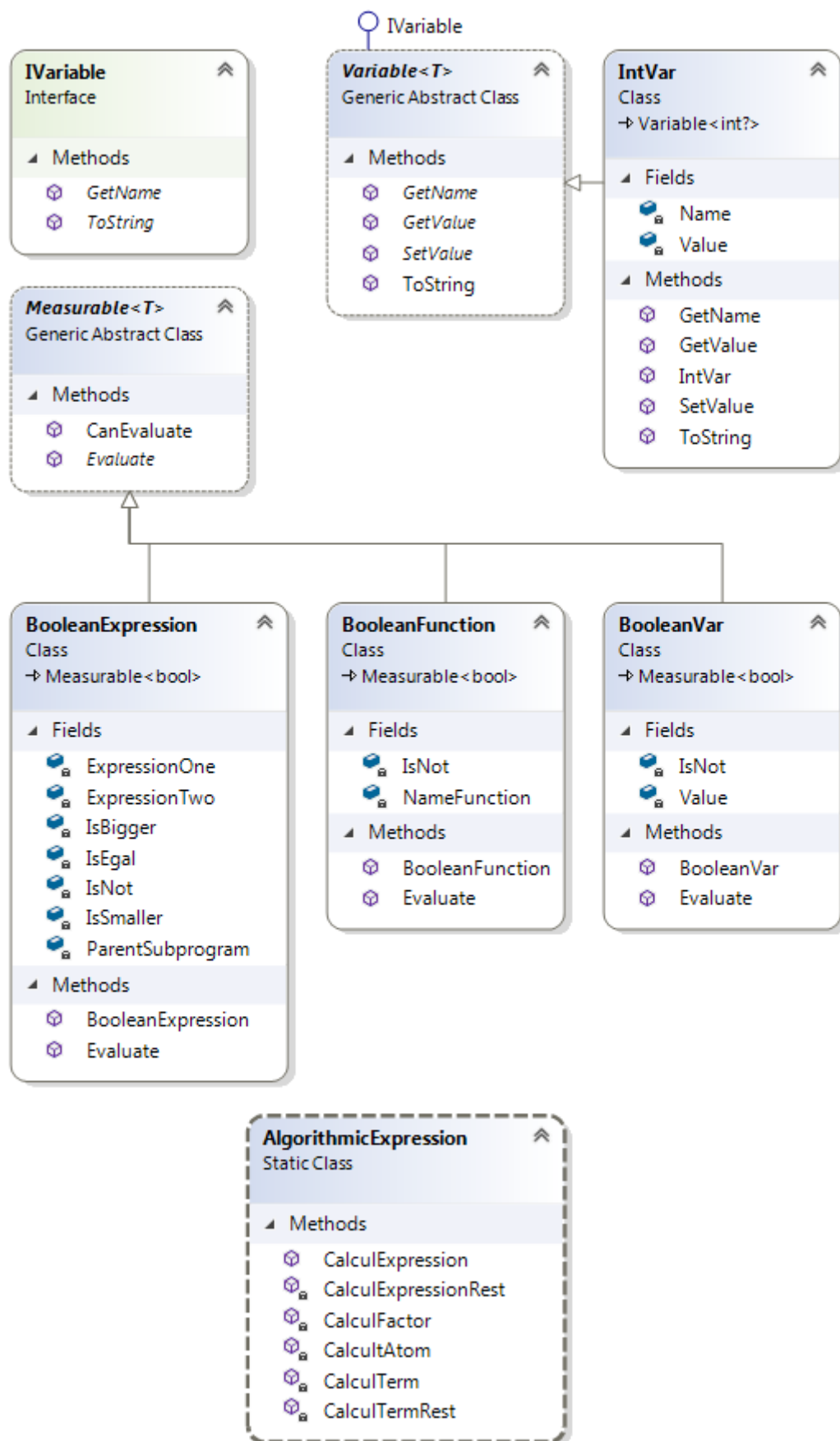
2.5.1.4 Les sept phéromones



Labyrinthe Couloir	
Code	Réaction
<pre> int NbPheromone = 0; int NbAvancer = 0; while(not NbPheromone = 7){ int NbAvancer = 0; if(SurUnePheromone){ PrendrePheromone; NbPheromone = NbPheromone + 1; Dire NbPheromone; } if(not NbPheromone = 7){ Avancer; NbAvancer = NbAvancer + 1; Dire NbAvancer; } } Dire 'fin du programme'; Dire NbPheromone; Dire NbAvancer; </pre>	<p>La fourmi ramasse les 7 phéromones puis s'arrête.</p> <p>Le terminal affiche :</p> <p>Fourmi dit : Variable nbavancer = 1 Fourmi dit : Variable nbavancer = 2 Fourmi dit : Variable nbavancer = 3 Fourmi dit : Variable nbpheromone = 1 Fourmi dit : Variable nbavancer = 4 Fourmi dit : Variable nbavancer = 5 Fourmi dit : Variable nbavancer = 6 Fourmi dit : Variable nbpheromone = 2 Fourmi dit : Variable nbavancer = 7 Fourmi dit : Variable nbavancer = 8 Fourmi dit : Variable nbavancer = 9 Fourmi dit : Variable nbpheromone = 3 Fourmi dit : Variable nbavancer = 10 Fourmi dit : Variable nbavancer = 11 Fourmi dit : Variable nbavancer = 1 Fourmi dit : Variable nbpheromone = 4 Fourmi dit : Variable nbavancer = 13 Fourmi dit : Variable nbavancer = 14 Fourmi dit : Variable nbavancer = 15 Fourmi dit : Variable nbpheromone = 5 Fourmi dit : Variable nbavancer = 16 Fourmi dit : Variable nbavancer = 17 Fourmi dit : Variable nbavancer = 18 Fourmi dit : Variable nbpheromone = 6 Fourmi dit : Variable nbavancer = 19 Fourmi dit : Variable nbavancer = 20 Fourmi dit : Variable nbavancer = 21 Fourmi dit : Variable nbpheromone = 7 Fourmi dit : fin du programme Fourmi dit : Variable nbavancer = 0 Fourmi dit : Variable nbpheromone = 7</p>

2.5.2 MLD





2.5.3 Construction de l'arbre d'exécution

Durant ce projet, le développement de la construction de l'arbre a été le challenge principal pour pouvoir créer un arbre dynamique peu importe la longueur du code donné. Pour se faire, M. Carrel m'a fourni un code de base contenant les classes principales pour la structure de l'arbre dans l'optique de partir sur une base propre qu'il pourra reprendre plus tard pour continuer le développement de Krop.

La première étape est d'analyser le code avec l'analyseur syntaxique pour obtenir l'arbre de l'analyseur.

Par exemple avec ce code :

```
if (true){
    avancer;
}
```

Nous obtiendrons cet arbre de l'analyseur :

```
program<2001>
  statement<2002>
    ifElseStatement<2003>
      ifStatement<2004>
        IF<1009>: "if", line: 1, col: 1
        conditonStatement<2008>
          LEFT_PAREN<1001>: "<", line: 1, col: 3
          conditionExpr<2009>
            conditionParameter<2010>
              TRUE<1007>: "true", line: 1, col: 4
              RIGHT_PAREN<1002>: ">", line: 1, col: 8
            LEFT_BRACE<1003>: "{", line: 1, col: 9
          statement<2002>
            instructionStatement<2007>
              INSTRUCTION<1013>: "avancer", line: 2, col: 2
              SEMICOLON<1005>: ";", line: 2, col: 9
            RIGHT_BRACE<1004>: "}", line: 3, col: 1
```

Cette arbre est composé de nœuds comme **program<2001>** qui contient un nœud enfant **statement<2002>**. Il contient aussi des tokens comme **INSTRUCTION<1013>** qui contient l'instruction écrite dans le code « **avancer** ». La construction de l'arbre d'exécution va être faite à l'aide de ces différents composants.

La deuxième étape consiste à créer un sous-programme avec en paramètre le nœud **program**. Ensuite le constructeur de la classe sous-programme va s'occuper de créer les différentes branches de ce nœud en regardant ces nœuds enfant.

Pseudo code du constructeur de la classe sous-programme :

Pour chaque nœud enfant du nœud

 Si égal à statement

 Pour chaque nœud enfant du statement

 Si égal à instructionStatement

 Crée une commande avec en paramètre le nœud enfant

 Si égal à ifElseStatement

 Crée une instruction if avec en paramètre le nœud enfant

 Si égal à whileStatement

 Crée une instruction while avec en paramètre le nœud enfant

Donc dans notre exemple, une instruction if sera créée. Puis dans le constructeur de la classe instructionIf, le nœud en paramètre sera traité de la même façon pour créer les branches du branchement conditionnel (if).

Pseudo code du constructeur de la classe instructionIf :

Pour chaque nœud enfant du nœud

Si égal à ifStatement

Crée un sous-programme avec en paramètre le nœud enfant

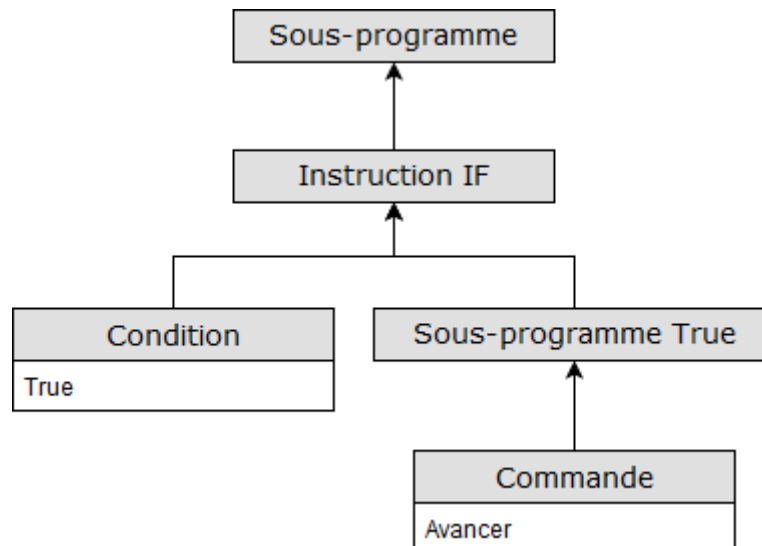
Enregistre la condition

Si égal à elseStatement

Crée un sous-programme avec en paramètre le nœud enfant

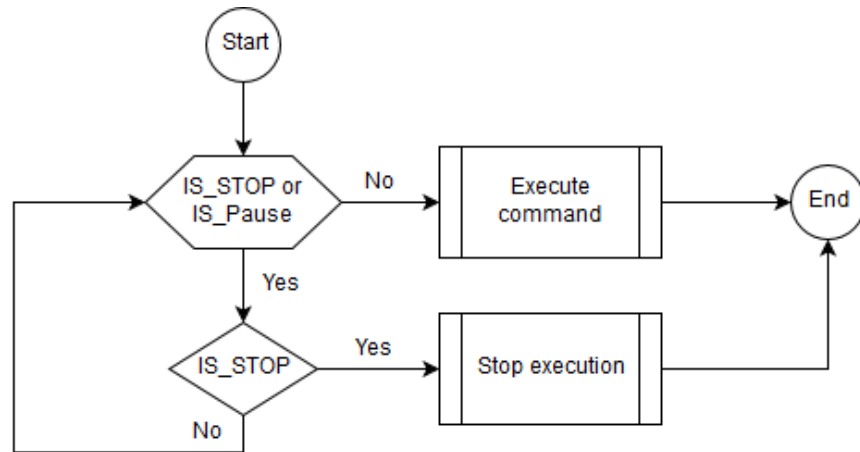
Un seul sous-programme pour la branche True du branchement conditionnel sera créé dans notre exemple. Le sous-programme créé agira comme expliqué au début et créera une commande « avancer ».

Au final, notre arbre d'exécution ressemblera à ceci :



2.5.4 Gestion des pauses et arrêts d'un programme en cours d'exécution

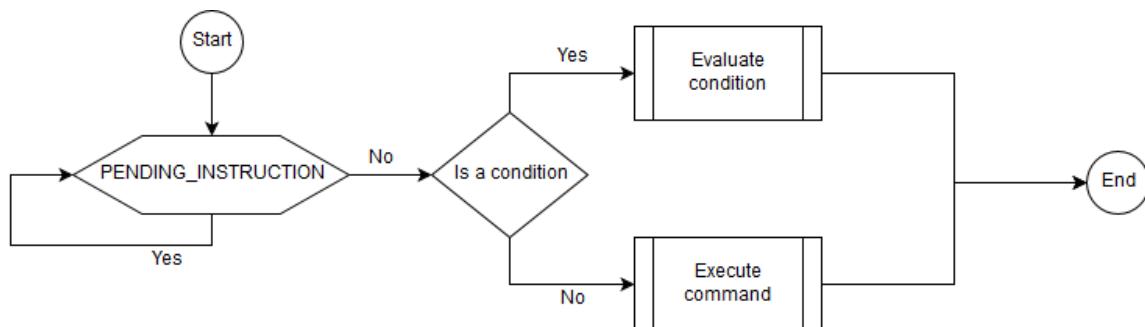
La gestion des pauses et arrêts d'un programme se fait à l'aide des boutons Stop, Pause et Play. Pour gérer cela, une vérification est faite avant chaque exécution de commande pour savoir si l'exécution est autorisée. Pour bloquer l'application lors d'une pause, une boucle while se lance continuellement tant que l'utilisateur n'a pas cliqué sur les boutons Play ou Stop. Le bouton Stop arrête l'exécution du programme.



2.5.5 Gestion du temps d'attente de la fin d'une animation

Pour pouvoir gérer le temps d'attente de la fin d'une animation, j'ai divisé Krop en deux Threads, un pour l'arbre d'exécution et la fenêtre de contrôle et le deuxième pour la fenêtre de Krohonde. Pour gérer ce temps d'attente, j'ai utilisé le même principe utilisé pour les pauses et les arrêts, à la seule différence que les conditions ont aussi une boucle d'attente.

Dès qu'une commande est exécutée, une variable globale booléenne nommée PENDING_INSTRUCTION passe à True. Puis à la fin de l'exécution de la commande, cette variable passe de nouveau à False et l'exécution du programme continue.



3 Réalisation

3.1 Dossier de réalisation

Krop et sa documentation se trouve sur ce répertoire GitHub :

- <https://github.com/Stugz52/Krop>

Dans la racine du GitHub, il y a :

- Le répertoire **Code** contient le source code de Krop
- Le répertoire **Documentation** contient le cahier des charges, la planification initiale, le journal de travail, le rapport de projet, le guide d'utilisation, la documentation technique et un répertoire Divers contenant des diagrammes, le fichier de grammaire, des maquettes et des scénarios
- Le répertoire **Krop** contient l'exécutable de la dernière version de Krop disponible et le répertoire Code contenant les programmes de l'utilisateur
- Le fichier **README.md** contient la description du TPI et le manuel d'installation

Une documentation technique est disponible dans le répertoire **Documentation** dans le dossier **Documentation Technique**.

Pour ce projet, j'ai eu besoin d'utiliser un analyseur syntaxique et lexicale. J'ai décidé d'utiliser un générateur de **Parser** (analyseur syntaxique et lexicale) nommé Grammatica. Ce dernier me permet de générer le code source en C# du Parser avec un fichier de grammaire **EBNF** modifié par Grammatica. Ce code source est ensuite intégré directement dans Krop. Pour utiliser ce Parser, j'ai ajouté la librairie de Grammatica au projet à l'aide du gestionnaire de package NuGet directement intégré dans Visual Studio.

Pour générer le **Parser**, j'ai utilisé le code source de Grammatica 1.6.0 en utilisant le fichier Java **grammatica-1.6.jar** et mon fichier de grammaire avec l'interpréteur de commandes Windows. Le fichier de grammaire est disponible sur le répertoire GitHub à cette adresse :

<https://github.com/Stugz52/Krop/tree/master/Documentation/Divers/Grammaire>

Pour effectuer ce projet, j'ai utilisé mon ordinateur du CPNV sous Windows 7 avec les logiciels suivant :

Tableau des logiciels	
Nom	Version
Excel	2016
GitHub Desktop	1.2.0
Paint.net	4.0.21
Tiled	1.1.3
Visual Studio Community	2017
Word	2016

Tableau des applications Web	
Nom	URL
Draw.io	https://www.draw.io/
MockFlow	https://www.mockflow.com
Trello	https://trello.com/

Pour développer Krop, j'ai utilisé les frameworks et librairies suivantes :

Tableau des frameworks et librairies	
Nom	Version
.Net Framework	4.6.1
OpenTK	2.0.0
PerCederberg.Grammatica	1.6.0

3.2 Description des tests effectués

3.2.1 Tests unitaires

Fenêtre de contrôle				
Scénario	17.5	25.5	31.5	6.6
Exécuter un programme sans fichier	OK	OK	OK	OK
Charger un programme	OK	OK	OK	OK
Exécuter un programme	OK	OK	OK	OK
Créer un programme	OK	OK	OK	OK
Sauvegarder un programme	OK	OK	OK	OK
Supprimer un programme	OK	OK	OK	OK
Stopper un programme		OK	OK	OK
Mettre en pause et reprendre un programme		OK	OK	OK

Arbre d'exécution			
Scénario	25.5	31.5	6.6
CodeFaux01	OK	OK	OK
CodeFaux02	OK	OK	OK
CodeFaux03	OK	OK	OK
CodeFaux04	OK	OK	OK
CodeFaux05	OK	OK	OK
CodeJuste01	OK	OK	OK
CodeJuste02	OK	OK	OK
CodeJuste03	OK	OK	OK
CodeJuste04	OK	OK	OK
CodeJuste05	OK	OK	OK
CodeJuste06	OK	OK	OK
CodeJuste07	OK	OK	OK
CodeJuste08	OK	OK	OK
CodeJuste09	OK	OK	OK
CodeJuste10	OK	OK	OK
CodeJuste11	OK	OK	OK
CodeJuste12	OK	OK	OK

Commandes de la Fourmi		
Scénario	31.5	6.6
PoserPheromone	OK	OK
PrendrePheromone	OK	OK
Avancer	OK	OK
TournerADroite	OK	OK
TournerAGauche	OK	OK
SurUnePheromone	OK	OK
ObstacleEnFace	OK	OK
ObstacleADroite	OK	OK
ObstacleAGauche	OK	OK

3.2.2 Tests d'intégration

Krop		
Scénario	31.5	6.6
Labyrinthe Couloir	OK	OK
Sept Pheromones		OK

3.2.3 Tests d'acceptation par M. Carrel

Fenêtre de contrôle				
Scénario	17.5	25.5	31.5	7.6
Exécuter un programme sans fichier	OK			
Charger un programme	OK			
Exécuter un programme	OK			
Créer un programme	OK			
Sauvegarder un programme	OK			
Supprimer un programme	OK			
Stopper un programme		OK		
Mettre en pause et reprendre un programme		OK		

Arbre d'exécution			
Scénario	25.5	31.5	7.6
CodeFaux01	OK		
CodeFaux02	OK		
CodeFaux03	OK		
CodeFaux04	OK		
CodeFaux05	OK		
CodeJuste01	OK		
CodeJuste02	OK		
CodeJuste03	OK		
CodeJuste04	OK		
CodeJuste05	OK		
CodeJuste06	OK		
CodeJuste07	OK		
CodeJuste08	OK		
CodeJuste09	OK		
CodeJuste10	OK		
CodeJuste11	OK		
CodeJuste12	OK		

Commandes de la Fourmi		
Scénario	31.5	7.6
PoserPheromone	OK	
PrendrePheromone	OK	
Avancer	OK	
TournerADroite	OK	
TournerAGauche	OK	
SurUnePheromone	OK	
ObstacleEnFace	OK	
ObstacleADroite	OK	
ObstacleAGauche	OK	

Krop		
Scénario	31.5	7.6
Labyrinthe Couloir	OK	OK
Sept Pheromones		OK

3.3 Erreurs restantes

Je n'ai trouvé aucune erreur dans mon application suite à mes nombreux tests.

3.4 Liste des documents fournis

Les documents fournis sont les fichiers se trouvant sur le répertoire GitHub comme décrit dans le chapitre « Dossier de réalisation ».

4 Conclusions

Le projet s'est déroulé sans aucun problème. J'ai respecté mes objectifs de Sprint tout au long de ce dernier. Ce projet étant une suite à mon Pré-TPI Krohonde donc je connaissais mon environnement de travail. Je suis satisfait de mon travail et je pense que je n'aurais pas pu faire plus dans le temps qui m'était donné. Au final, la version finale de Krop est fonctionnelle et peut être utilisée pour apprendre à programmer.

La majorité de mes objectifs ont été atteints. J'ai décidé avec M. Carrel de faire en priorité les points techniques évalués spécifiques au projet. Krop permet de faire des branchements conditionnels, des boucles While, des variables et des évaluations d'expressions. Je n'ai pas fait les boucles For et les fonctions avec et sans paramètres. La fourmi peut avancer, tourner à droite, tourner à gauche, détecter des obstacles à proximité, détecter une phéromone, prendre une phéromone, poser une phéromone et parler. La commande permettant de se mettre en mouvement dans une direction donnée par un point cardinal n'a pas été faite car M. Carrel n'en voyait pas le besoin. Et la fourmi ne peut pas demander de valeur à introduire à l'utilisateur.

Grâce à ce TPI, j'ai pu découvrir de nouvelles choses que je ne connaissais comme par exemple la gestion de deux Threads et l'analyse syntaxique et lexicale. J'ai aussi pu pour la première fois développer une application complète en C#. J'ai pu m'améliorer globalement sur la gestion de projet et la programmation en C# avec une librairie graphique. J'aurais aimé avoir plus de temps pour pouvoir encore faire plus de fonctionnalité à Krop et améliorer son aspect visuel.

J'ai eu des difficultés à gérer plusieurs Threads en même temps et à utiliser un analyseur de syntaxe comme Grammatica car c'était la première fois que j'utilisais ces outils. Pour résoudre ce problème, j'ai dû passer par une phase d'apprentissage qui m'a pris beaucoup de temps sur mon TPI.

Krop peut être encore grandement amélioré. Il faudrait d'abord rajouter les fonctionnalités restantes du cahier des charges. Ensuite, il serait possible de rajouter un éditeur de jardins permettant de cliquer sur l'écran pour changer le type de la case ou pour poser un élément comme une phéromone ou la fourmi. Cela permettrait d'éviter à l'utilisateur d'aller modifier un fichier texte dans un éditeur de texte. Il est bien sûr possible de rajouter de nouvelles fonctionnalités à la fourmi. Une amélioration des graphismes serait une bonne chose pour rendre Krop plus attrayant.

5 Annexes

5.1 Résumé du rapport du TPI / version succincte de la documentation

Situation de départ

Krop a pour but de remplacer l'outil d'apprentissage à la programmation utilisé par les élèves de première année en utilisant un langage de programmation propre à Krop.

Au début de mon TPI, j'avais déjà mon code du Pré-TPI Krohonde pour l'aspect visuel de l'application et M. Carrel m'a fourni un code de base pour développer l'arbre d'exécution. Je n'avais aucune connaissance à propos des analyseur syntaxique et l'utilisation de plusieurs Threads simultanément.

Mise en œuvre

Avec M. Carrel, nous avons décidé de découper mon projet en quatre Sprint ayant chacun un objectif de développement. Le premier Sprint avait pour but de créer la fenêtre de contrôle. Le deuxième Sprint avait comme thème le développement de l'arbre d'exécution en utilisant l'analyseur syntaxique Grammatica. Le troisième Sprint avait comme objectif la programmation des différentes actions de la fourmi et le dernier Sprint devait permettre d'implémenter les variables, les expressions algorithmiques et l'évaluation expressions booléennes.

Pour pouvoir atteindre les objectifs de mon TPI, j'ai dû apprendre à utiliser Grammatica, à écrire une application complète en C# et à utiliser différents Threads pour effectuer ce TPI.

J'ai dû créer une stratégie de test complète pour créer une application résistante au programme de l'utilisateur. J'ai écrit des scénarios couvrant toutes les fonctionnalités de Krop pour pouvoir tester chaque cas. J'ai effectué des tests à chaque Sprint et j'ai validé ces derniers avec M. Carrel lors des Sprint review.

Résultats

Au final, l'application Krop est fonctionnelle et permet d'effectuer des programmes simples malgré le fait que tous les objectifs du cahier des charges n'ont pas été accomplis. Je pense que cette application pourra être utilisé à la rentrée prochaine après une vérification et amélioration de la part de M. Carrel.

5.2 Glossaire

Mot	Définition
Grammatica	Grammatica est un générateur de Parser. Pour plus d'info, voici le site de ce projet : https://grammatica.percederberg.net/index.html
EBNF	EBNF (Extended Backus-Naur Form) est un langage de grammaire. Pour plus d'info, vous trouverez une explication sur ce site : http://www.garshol.priv.no/download/text/bnf.html

5.3 Historique des modifications

	Sprint 1	Sprint 2	Sprint 3	Sprint 4
1.1 Introduction	Ajouté			
1.2 Objectifs	Ajouté			
1.3 Planification initiale	Ajouté			
2.1.1 Vue d'ensemble	Ajouté	Modifié	Modifié	
2.1.2 MCD		Ajouté		
2.2 Stratégie de test		Ajouté		
2.3 Risques techniques		Ajouté		
2.4 Planification		Ajouté	Modifié	Modifié
2.5.1 Maquettes / Use cases / Scénarios	Ajouté	Modifié		Modifié
2.5.2 MLD		Ajouté		Modifié
2.5.3 Construction de l'arbre d'exécution		Ajouté	Modifié	
2.5.4 Gestion des pauses et arrêts d'un programme en cours d'exécution			Ajouté	
2.5.5 Gestion du temps d'attente de la fin d'une animation			Ajouté	
3.1 Dossier de réalisation		Ajouté	Modifié	Modifié
3.2.1 Tests unitaires	Ajouté	Modifié	Modifié	Modifié
3.2.2 Tests d'intégration		Ajouté	Modifié	Modifié
3.2.3 Tests d'acceptation par M. Carrel		Ajouté	Modifié	Modifié
3.3 Erreurs restantes				Ajouté
3.4 Liste des documents fournis				Ajouté
4 Conclusions				Ajouté
5.1 Résumé du rapport du TPI / version succincte de la documentation				Ajouté
5.2 Glossaire			Ajouté	
5.3 Historique des modifications		Ajouté	Modifié	Modifié
5.4 Sources – Bibliographie		Ajouté		Modifié
5.5 Journal de bord		Ajouté		Modifié
5.5 Manuel d'Installation			Ajouté	Modifié
5.6 Manuel d'Utilisation				Ajouté
5.7 Archives du projet				Ajouté

5.4 Sources – Bibliographie

Les sites internet :

- <https://tomassetti.me/parsing-in-csharp/>
- <https://tomassetti.me/guide-parsing-algorithms-terminology/>
- <https://grammatica.percederberg.net/index.html>
- https://fr.wikipedia.org/wiki/Symboles_terminaux_et_non_terminaux
- <https://grammatica.percederberg.net/doc/manual/index.html>
- <http://www.garshol.priv.no/download/text/bnf.html>
- https://softwareengineering.stackexchange.com/questions/254074/how-exactly-is-an-abstract-syntax-tree-created?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa
- <https://msdn.microsoft.com/en-us/library/>

Les aides externes :

- Xavier Carrel, le chef de projet

5.5 Journal de bord

Date	Événement
08.05.2018	Signature du cahier des charges avec Bertrand Sahli
08.05.2018	Sprint 1 planning avec Xavier Carrel
17.05.2018	Sprint 1 review avec Xavier Carrel
17.05.2018	Sprint 2 planning avec Xavier Carrel
22.05.2018	Visite du TPI avec Nicolas Tieche
25.05.2018	Sprint 2 review avec Xavier Carrel
25.05.2018	Sprint 3 planning avec Xavier Carrel
31.05.2018	Sprint 3 review avec Xavier Carrel
31.05.2018	Sprint 4 planning avec Xavier Carrel
07.06.2018	Sprint 4 review avec Xavier Carrel
07.06.2018	Rendu du TPI

Le journal de travail se trouve en format Excel sur le répertoire GitHub à cette adresse :
<https://github.com/Stugz52/Krop/blob/master/Documentation/Journal%20de%20travail%20-%20SGZ.xlsm>

5.6 Manuel d'Installation

Pour installer l'application Krop, il faut :

1. Cloner le fichier Krop/Krop.zip du répertoire GitHub en local
2. Dézipper le fichier Krop.zip

5.7 Manuel d'Utilisation

Krop est un projet C# permettant d'aborder les notions de bases de la programmation en écrivant un programme simple dans un langage de programmation simplifié qui dicte le comportement d'une fourmi.

5.7.1 Prérequis

Pour pouvoir utiliser Krop, il vous faut :

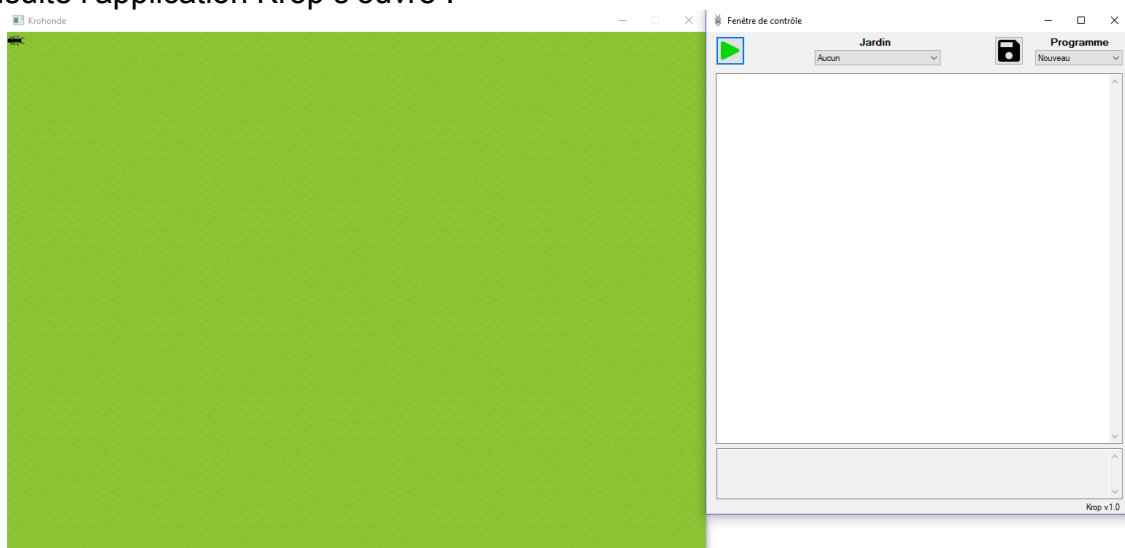
- Utiliser Windows 7 ou 10
- Faire une copie locale du dossier « Krop »

5.7.2 Démarrage

Pour démarrer Krop, il faut procéder comme suit :

- Ouvrez le dossier « Krop » (la copie locale)
- Double-cliquez sur le fichier « Krop.exe »

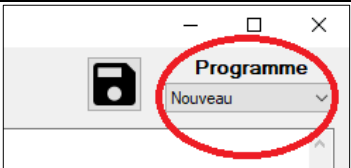
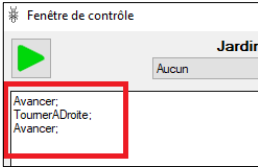
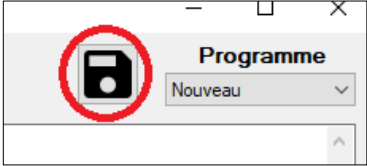
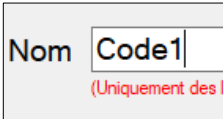
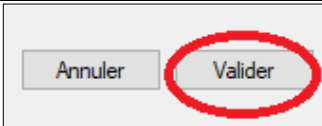
Ensuite l'application Krop s'ouvre :






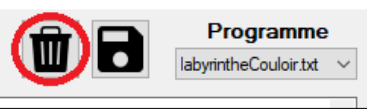
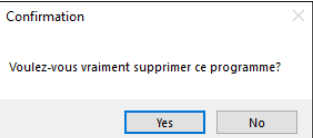
L'application est composée des deux fenêtres suivantes :


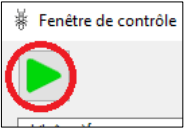
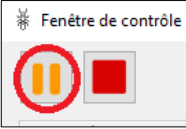
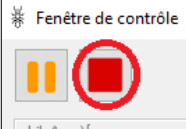
- La fenêtre Krohonde affiche les résultats de l'exécution du programme
- La fenêtre de contrôle permet de créer, modifier ou supprimer un programme. Elle permet aussi de démarrer, mettre en pause ou stopper un programme. Le terminal en bas de la fenêtre affiche les messages d'erreur et les paroles de la fourmi.

5.7.3 La fenêtre de contrôle

Créer un nouveau programme		
1	Sélectionnez « Nouveau » dans la liste des programmes	
2	Écrivez votre code dans le champ texte	
3	Cliquez sur le bouton « sauvegarder »	
4	Écrivez le nom de votre nouveau programme Attention si vous réutilisez le même nom qu'un programme déjà existant, ce dernier sera remplacé par le nouveau	
5	Cliquez sur le bouton « Valider »	

Sauvegarder un programme		
1	Sélectionnez votre programme dans la liste des programmes	
2	Modifiez votre code	
3	Cliquez sur le bouton « Sauvegarder »	









Supprimer un programme		
1	Sélectionnez votre programme dans la liste des programmes	
2	Cliquez sur le bouton « Supprimer »	
3	Cliquez sur le bouton « Yes »	

Changer de jardin		
1	Sélectionnez votre jardin dans la liste des jardins	
Démarrer un programme		
1	Cliquez sur le bouton « Démarrer »	
Mettre en pause un programme		
1	Cliquez sur le bouton « Pause »	
Stopper un programme		
1	Cliquez sur le bouton « Stop »	

5.7.4 Les jardins

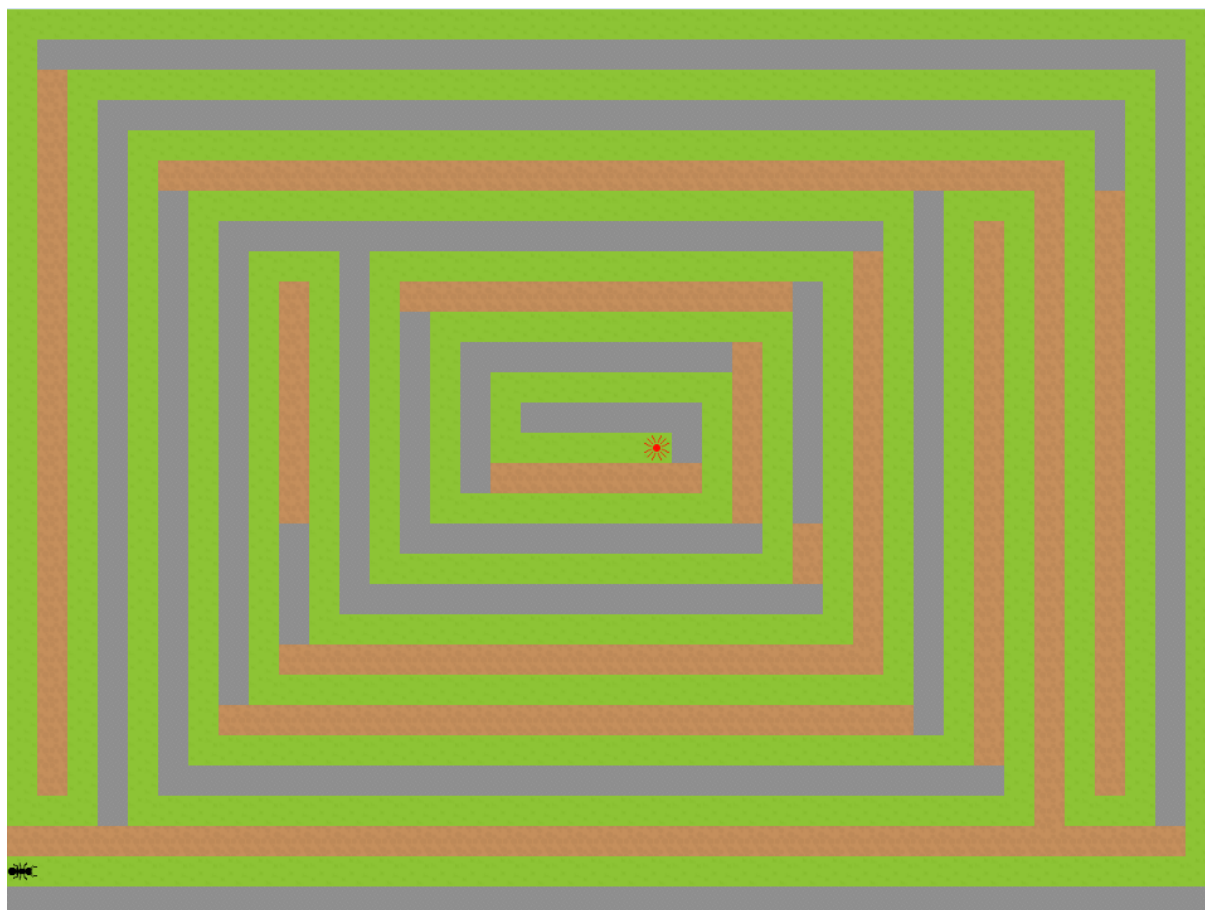
Un jardin est un fichier **.txt** se trouvant dans le dossier « Garden » à la racine du dossier « Krop ». Le fichier doit contenir 30 lignes contenant chacune 40 caractères autorisés.

[illegible]

Caractères autorisés		
.	Herbe	
A	Fourmilière (Obstacle)	
R	Roche (Obstacle)	
P	Phéromone	
N	Fourmi en direction du nord	
E	Fourmi en direction de l'est	
S	Fourmi en direction du sud	
W	Fourmi en direction de l'ouest	

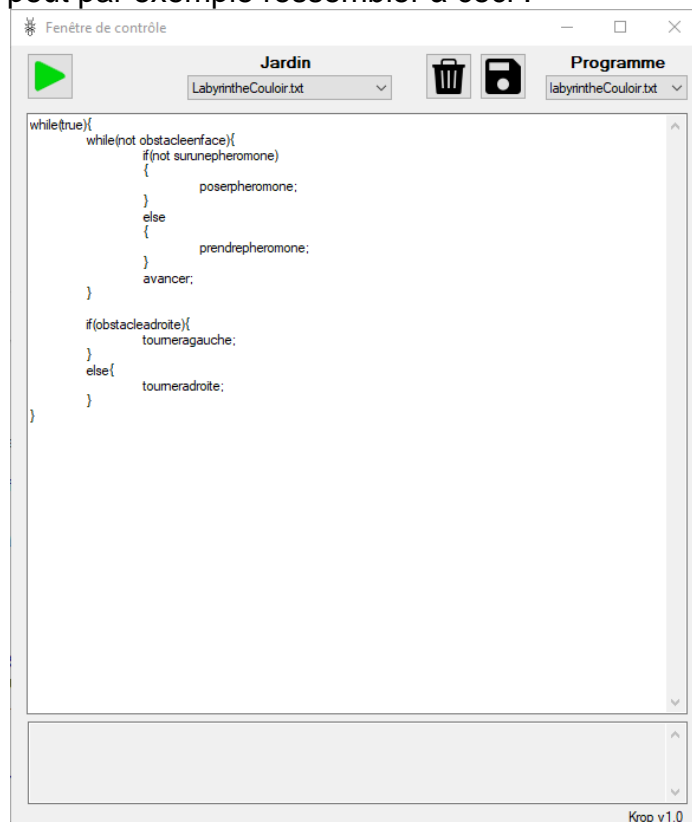
Labyrinth.txt

Le résultat du fichier labyrinthe.txt :



5.7.5 Les programmes

Un programme est fichier **.txt** se trouvant dans le dossier « Code » à la racine du dossier « Krop ». Il peut être composé de tous les éléments se trouvant dans la partie « Syntaxe de Krop ». Un programme permettant de résoudre le labyrinthe du jardin « Labyrinthe.txt » peut par exemple ressembler à ceci :



5.7.6 Syntaxe de Krop

Glossaire	
Condition	Ce terme doit être remplacé par une instruction du tableau « Les conditions »
<i>Déclaration</i>	Ce terme doit être remplacé par une instruction du tableau « Les déclarations de variables »
<u>Instruction</u>	Ce terme doit être remplacé par une instruction du tableau « Les instructions »
NomVariable	Ce terme doit être remplacé par le nom de la variable qui doit être uniquement composé de lettre
<u>Expression</u>	Ce terme désigne une expression algorithmique pouvant contenir des chiffres et des variables Int
NOT	Inverse le résultat de la condition
*	Le symbole « * » signifie que le terme à gauche peut apparaître de 0 à plusieurs fois
+	Le symbole « + » signifie que le terme à gauche peut apparaître de 1 à plusieurs fois
?	Le symbole « ? » signifie que le terme à gauche peut apparaître de 0 à 1 fois

Les instructions de base		
IF	<pre> If(NOT? Condition) { Déclaration* Instruction+ } Else { Déclaration* Instruction+ } </pre>	L'instruction If n'est pas obligée d'avoir une branche « Else » et le NOT n'est pas obligatoire
WHILE	<pre> While(NOT? Condition) { Déclaration* Instruction+ } </pre>	Le NOT n'est pas obligatoire

Les déclarations de variables		
INT	<pre> Int NomVariable = <u>Expression</u> ; </pre>	Attention une déclaration de variable ne peut qu'être déclaré en début de programme, While, If et Else

Les conditions	
True	Retourne True
False	Retourne False
ObstacleEnFace	Retourne True s'il y a un obstacle en face de la fourmi
ObstacleADroite	Retourne True s'il y a un obstacle à droite de la fourmi
ObstacleAGauche	Retourne True s'il y a un obstacle à gauche de la fourmi
SurUneGraine	Retourne True s'il y a une phéromone en dessous de la fourmi
<u>Expression</u> = <u>Expression</u>	Retourne True si les deux expressions algorithmiques sont égales
<u>Expression</u> > <u>Expression</u>	Retourne True si la première expression est plus grande que la seconde
<u>Expression</u> < <u>Expression</u>	Retourne True si la première expression est plus petite que la seconde

Les instructions	
Avancer ;	La fourmi avance d'une case devant elle
TournerADroite ;	La fourmi pivote de 90° vers la droite
TournerAGauche ;	La fourmi pivote de 90° vers la gauche
PoserPheromone ;	La fourmi pose une phéromone sous elle
PrendrePheromone ;	La fourmi prend une phéromone sous elle
Dire <i>NomVariable</i> ;	Affiche dans le terminal la valeur de la variable
Dire 1234 ;	Affiche dans le terminal le nombre écrit après l'instruction Dire
Dire 'Hello World' ;	Affiche dans le terminal le texte écrit entre les « ' » Attention uniquement des lettres, chiffres et espaces
<i>NomVariable</i> = <u>Expression</u> ;	Change la valeur de la variable avec le résultat de l'expression

5.8 Archives du projet

- Trois CD contenant les fichiers du répertoire GitHub
- Trois versions papier du rapport de projet et du journal de travail