SUPINFO Academic Dept.

SUPINFO International University

INSTITUTE OF INFORMATION TECHNOLOGY

UML et programmation en C++

Mini-projet

La fourmi de Langton

Version 1.0
Last update: 19/12/2013
Use: Students/Staff
Author: Laurent Godefroy

SOMMAIRE

1 PREAMBULE

Cet examen est bien sûr **individuel**. Toute forme de plagiat ou utilisation de codes disponibles sur internet ou tout autre support, même de manière partielle, est strictement interdite et se verra sanctionnée d'un 0, d'une mention « cheater », et le cas échéant d'un conseil de discipline.

Vous devrez envoyer les codes sources de votre projet par mail à votre formateur avant le dimanche 19 janvier à 23h59, heure locale. Au delà de cette date et heure votre note sera de 0. Vous comprimerez ces codes sources dans une archive au format « .zip ».

Ce mini-projet donne lieu à des soutenances qui se dérouleront la semaine du 20 Janvier.

Les soutenances sont également individuelles. Elles dureront **20 minutes** pendant lesquelles vous montrerez à votre examinateur le bon fonctionnement de votre programme en en faisant la démonstration. Si vous n'avez pas implémenté le projet dans sa totalité, vous exposerez les parties fonctionnelles.

Pour appuyer votre présentation, vous devrez préparer un fichier de type Powerpoint, dans lesquels vous expliquerez les points du code que vous jugez les plus importants et significatifs. Il n'est pas nécessaire d'envoyer ce fichier à votre examinateur, ce dernier le découvrira le jour de la soutenance. Une communication précisant tout cela vous sera envoyé début janvier.

Un barème **indicatif** vous est donné dans la dernière partie de ce sujet.

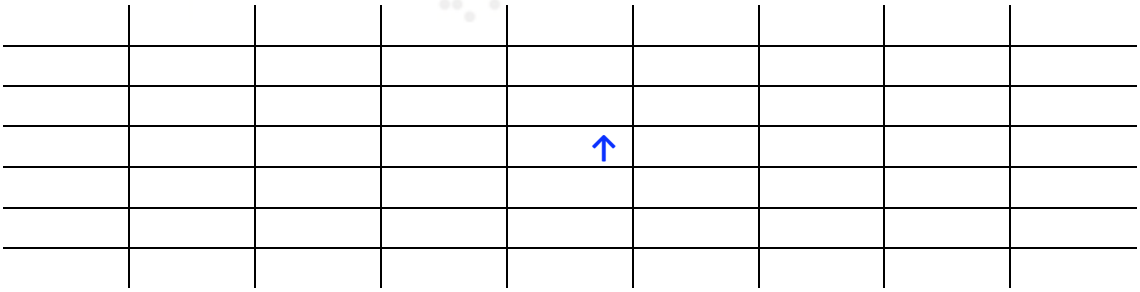
2 GENERALITES SUR CE PROJET

Remarque importante : aucun code n'est demandé dans cette partie qui n'est qu'explicative.

Le but de ce projet est d'implémenter en C++ un automate cellulaire inventé en 1986 par Chris Langton. On peut se l'imaginer comme étant l'étude du mouvement d'une fourmi sur un quadrillage infini, dont les cases peuvent être de deux couleurs, blanches ou noires. Bien sûr ses déplacements ne sont pas aléatoires et sont soumis à des règles très précises. Typiquement, la façon de se mouvoir de la fourmi dépendra de la couleur de la case où elle se trouve, et de son orientation (nord, ouest, est ou sud) actuelle. On étudiera également une de ses généralisations : la turmite.

2.1 AUTOMATES CELLULAIRES DU TYPE DE LA FOURMI DE LANGTON

Imaginons donc une grille où initialement toutes les cases sont blanches. On va représenter la fourmi par une flèche indiquant sa direction. Initialement, on place une fourmi « au milieu », et on lui attribue une orientation « vers le nord » :

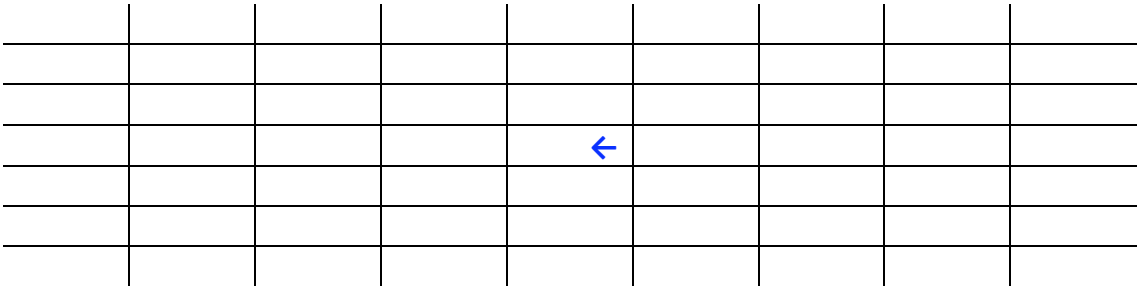


A chaque itération, la fourmi va se déplacer d'une case en suivant le même processus :

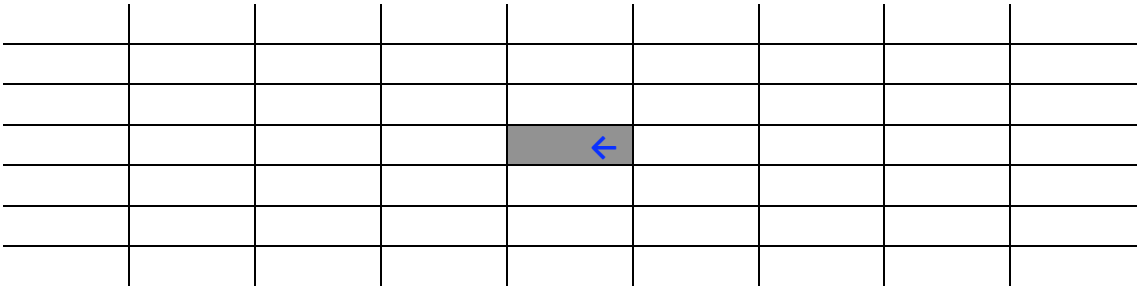
1. Si elle est sur case blanche elle tourne de 90° vers la gauche, si elle est sur une case noire elle tourne de 90° vers la droite.
2. Elle inverse la couleur de la case où elle se trouve (blanche ↔ noire).
3. Elle se déplace d'une case dans la direction de son orientation.

Pour le premier déplacement de notre fourmi, cela donne donc cela :

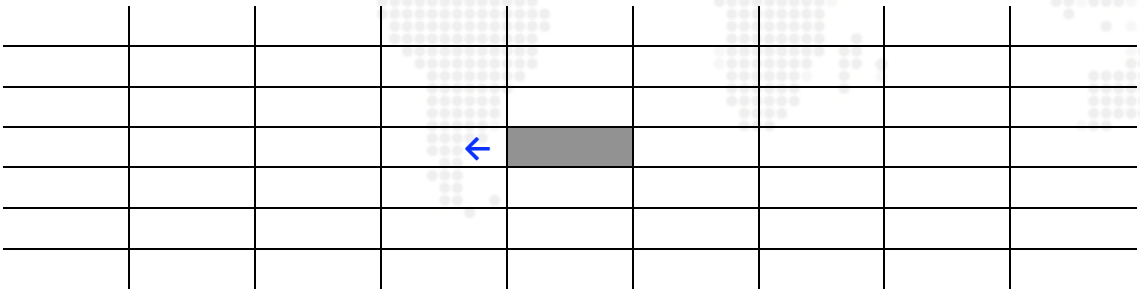
1. Elle est sur une case blanche donc tourne de 90° vers la gauche :



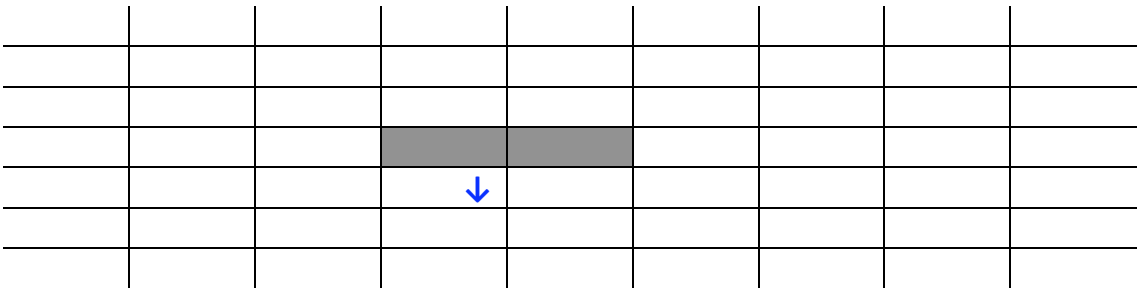
2. Elle inverse la couleur de la case où elle se trouve, en l'occurrence passage du blanc au noir :



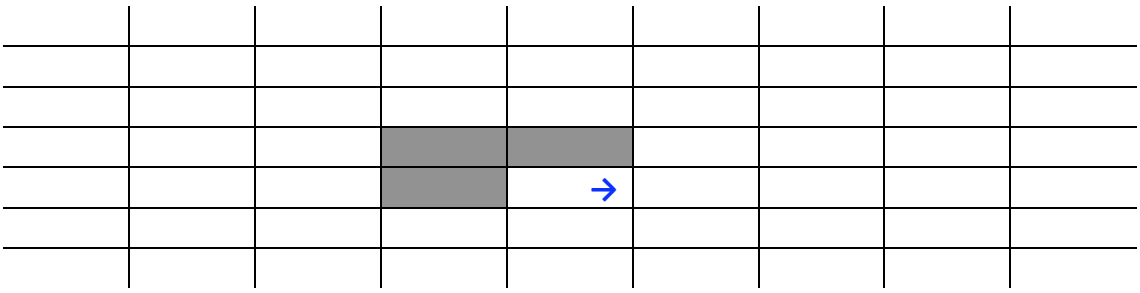
3. Elle se déplace d'une case dans la direction de son orientation, en l'occurrence « ouest » :



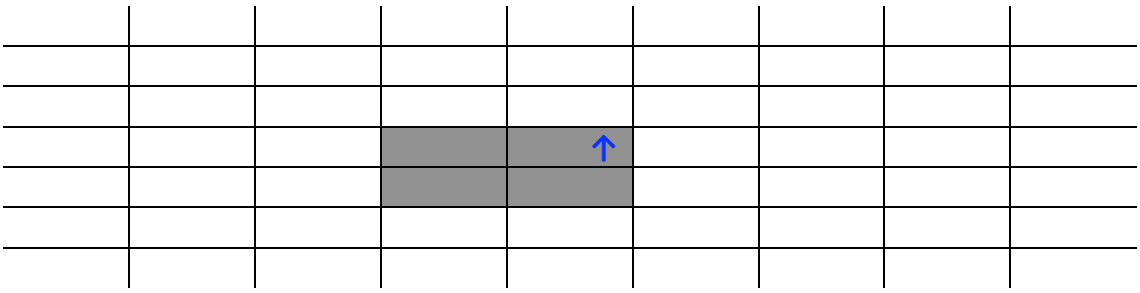
Après les trois étapes du deuxième déplacement, on obtiendra :



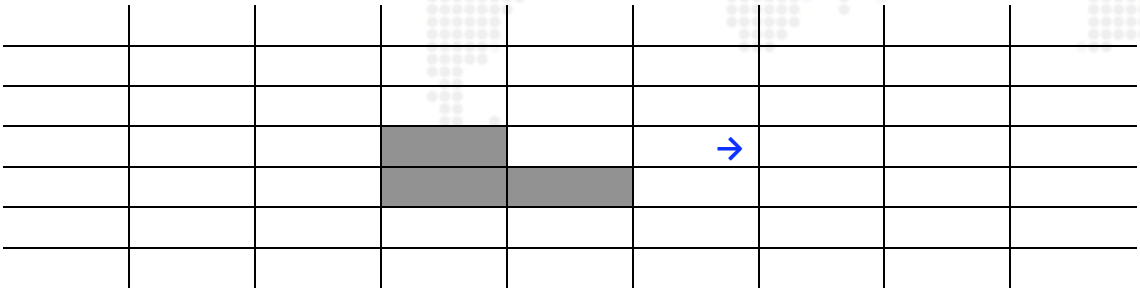
Après les trois étapes du troisième déplacement, on obtiendra :



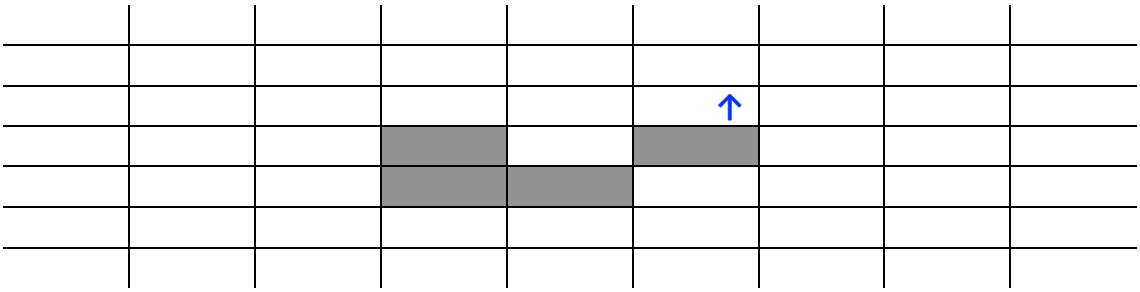
Après les trois étapes du quatrième déplacement, on obtiendra :



Après les trois étapes du cinquième déplacement, on obtiendra :



Après les trois étapes du sixième déplacement, on obtiendra :



Et ainsi de suite….

Sur le même modèle, on pourra nous même imposer :

- la façon dont notre fourmi change de direction selon la couleur de la case où elle se trouve.
- le changement de couleur de la case qu'elle quitte.

Avec des notations évidentes ('G' pour 'gauche', 'D' pour 'droite', 'B' pour 'blanche' et 'N' pour 'noire') la fourmi de Langton peut donc être décrite par :

B		N	
N	G	B	D

2.2 UNE GENERALISATION : LES TURMITES

Les turmites sont des généralisations de la fourmi de Langton. Le principe général reste le même, mais cette fois les fourmis possèdent deux états.

Au processus de déplacement d'une fourmi s'ajoute donc une quatrième phase, ici mise en évidence avec des caractères gras :

1. Si elle est sur case blanche elle tourne de 90° vers la gauche ou la droite, si elle est sur une case noire elle tourne de 90° vers la gauche ou la droite.
2. Elle inverse ou pas la couleur de la case où elle se trouve (une inversion est du type : blanche ↔ noire).
3. Elle se déplace d'une case dans la direction de son orientation.
4. **Elle inverse ou pas l'état dans laquelle elle se trouve (une inversion est du type : état1 ↔ état2).**

Si aux notations précédentes ('G' pour 'gauche', 'D' pour 'droite', 'B' pour 'blanche' et 'N' pour 'noire') on ajoute '1' pour le premier état et '2' pour le second, une turmite pourra donc être décrite comme cela :

	B			**N**		
1	N	G	2	B	G	2
2	N	D	2	B	G	1

Remarque : le tableau ci-dessus n'est qu'un exemple et ne décrit qu'une des 2^{12} turmites existantes (on a en effet 2 valeurs possibles pour chacune des douze cases de ce tableau).

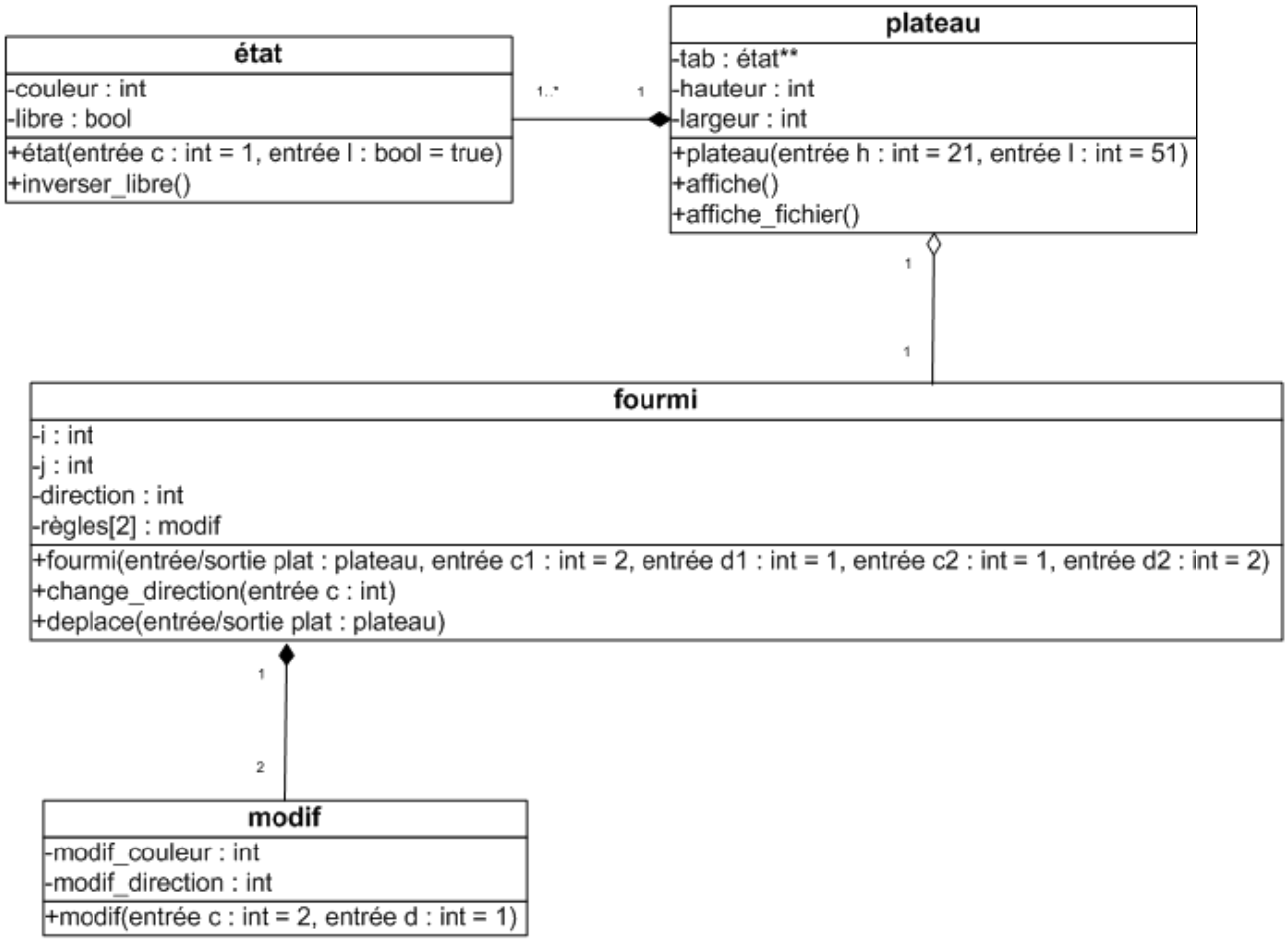
3 PROGRAMMATION DES AUTOMATES CELLULAIRES DU TYPE DE LA FOURMI DE LANGTON

Il vous est fortement recommandé de lire l'intégralité de cette partie avant de commencer à coder. Les travaux demandés sont mis en évidence avec une couleur bleue.

3.1 DIAGRAMME DE CLASSES

On commence par donner un diagramme de classes qui sera implémenté lors des deux prochaines sous-parties.

Remarque importante : dans le diagramme suivant et dans la suite de l'énoncé on ne fait figurer aucun « getter » ni « setter ». Cela sera à vous de les ajouter quand le besoin s'en fera sentir. En faisant en sorte pour les « setter » qu'ils respectent les mêmes éventuelles contraintes sur les attributs que les constructeurs.



3.2 GESTION DE LA GRILLE

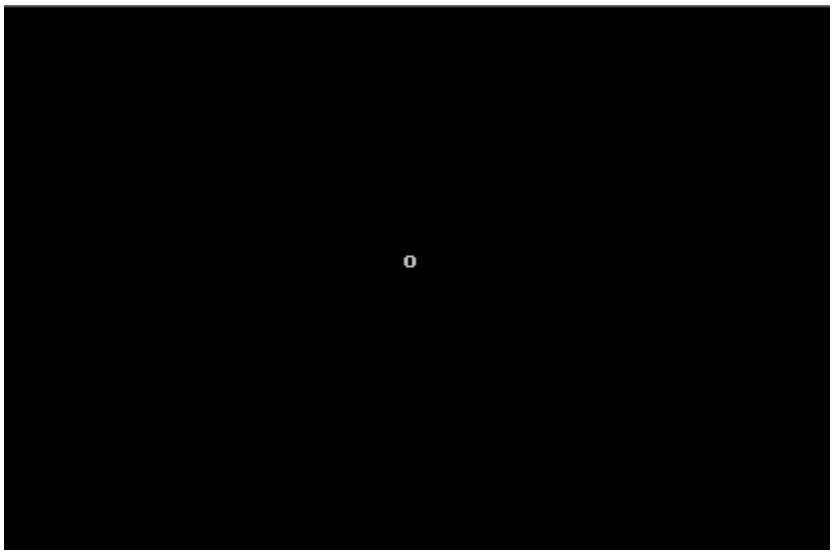
Déclarer dans un « .h » et implémenter dans un « .cpp » les classes « état » et « plateau ».

Quelques petites précisions sur la classe « état » :

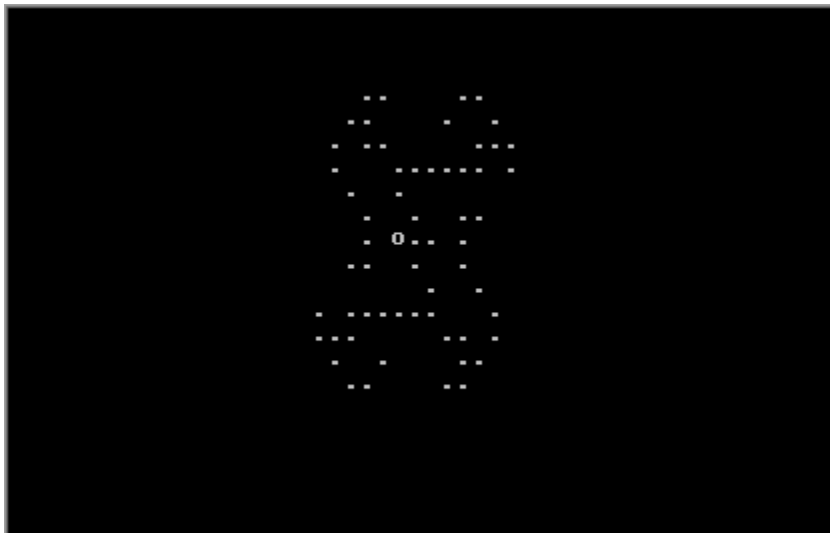
- Cette classe sert à décrire une case, l'attribut « libre » indique donc si elle est occupée ou non, et l'attribut « couleur » vaut 1 si la case est blanche et 2 si elle est noire. On fera en sorte que cet attribut ne puisse pas prendre d'autres valeurs.
- La méthode « inverser_libre » change l'occupation de la case.

Quelques petites précisions sur la classe « plateau » :

- « hauteur » et « largeur » désignent respectivement le nombre de lignes et de colonnes du quadrillage. Ils sont donc nécessairement positifs. On exige en plus qu'ils soient impairs sinon on les remplace par leurs valeurs par défaut.
- « tab » désigne le quadrillage en lui même, il s'agit donc d'un tableau d'objets de la classe « état ».
- Le constructeur de cette classe dimensionne le plateau, chaque case étant initialement libre et de couleur blanche.
- La méthode « affiche » réalise l'affichage du plateau sur la console avec les conventions suivantes : une case libre et blanche est représentée par un ' ' (espace), une case libre et noire par un '.' et une case occupée par un 'o' quelque soit sa couleur (et dans la partie 4 quelque soit l'état de la fourmi).
  Cela donne donc cela à la création du plateau :



Et après quelques générations :



- La méthode « affiche_fichier » réalise la même chose mais cette fois dans un fichier texte (un léger travail de documentation est donc nécessaire pour implémenter cette méthode).

Autre travail à réaliser : mettre cette classe sous sa forme canonique de Coplien.

Dernière chose à faire pour cette partie : Surcharger l'opérateur « ++ » version préfixe afin qu'il augmente de deux à la fois la hauteur et la largeur du quadrillage. On conservera les états antérieurs des cases, les nouvelles étant initialisées à « true » et 1. On fera en sorte d'ajouter une ligne en dessus et en dessous de celles déjà existantes, et d'ajouter une colonne à gauche et à droite de celles déjà existantes.

Remarque : la mise sous forme canonique de Coplien et la surcharge de l'opérateur « ++ » n'apparaissent pas dans le diagramme de classes ci-dessus.

3.3 GESTION D'UNE FOURMI

Déclarer dans « .h » et implémenter dans un « .cpp » les classes « modif » et « fourmi ».

Quelques précisions sur la classe « modif » :

- Cette classe sert à indiquer quelles modifications seront effectuées sur la couleur d'une case et sur l'orientation d'une fourmi lors d'un déplacement de cette dernière. Mais elle n'effectue pas ces modifications.
- L'attribut « modif_couleur » vaut 1 ou 2 et indique simplement quelle sera la nouvelle couleur de la case quittée.
- L'attribut « modif_direction » vaut 1 pour indiquer une rotation de 90° degrés vers la gauche, et 2 pour une indiquer une rotation de 90° degrés vers la droite.

Quelques précisions sur la classe « fourmi » :

- La classe « fourmi » s'occupe elle de gérer tout ce qui concerne notre fourmi.
- Les attributs « i » et « j » repèrent sa position sur le plateau.
- L'attribut « direction » est un entier valant 1, 2, 3 ou 4 (respectivement pour 'nord', 'ouest', 'sud' et 'est'). Il ne peut pas prendre d'autres valeurs.
- « règles » est un tableau de deux objets de la classe « modif », le premier indiquant les changements lorsque notre fourmi est une sur une case blanche, et le second lorsqu'elle est sur une case noire.
  Par exemple, pour la fourmi de Langton les attributs « modif_couleur » et « modif_direction » du premier objet valent respectivement 2 et 1, et ceux du second objet 1 et 2.
- Le constructeur de cette classe, fait en sorte que la fourmi soit positionnée initialement « au milieu » du plateau avec une direction vers le nord. Elle initialise les valeurs de « règles » avec les paramètres c1, d1, c2, d2.

- La méthode « change_direction » prend en paramètre un entier supposé valoir 1 ou 2 et réalise une rotation de 90° degrés vers la gauche s'il vaut 1, et une rotation de 90° degrés vers la droite s'il vaut 2. Cette méthode modifie donc l'attribut « direction ».
- La méthode « déplace » réalise le déplacement d'une fourmi en suivant le processus en 3 étapes décrit dans la partie « Généralités ».
- Si la fourmi se rapproche trop du bord on agrandira le plateau à l'aide de la surcharge de l'opérateur ++.

3.4 UN PROGRAMME POUR FAIRE BOUGER NOTRE FOURMI

Dans le « main », on déclarera un plateau, et on affichera les N premiers déplacements de la fourmi, N étant une valeur saisie par l'utilisateur. On pourra effacer la console entre deux affichages.

On utilisera également un affichage dans un fichier texte pour déclarer un plateau assez conséquent pour pouvoir simuler quelques milliers de déplacements de la fourmi.

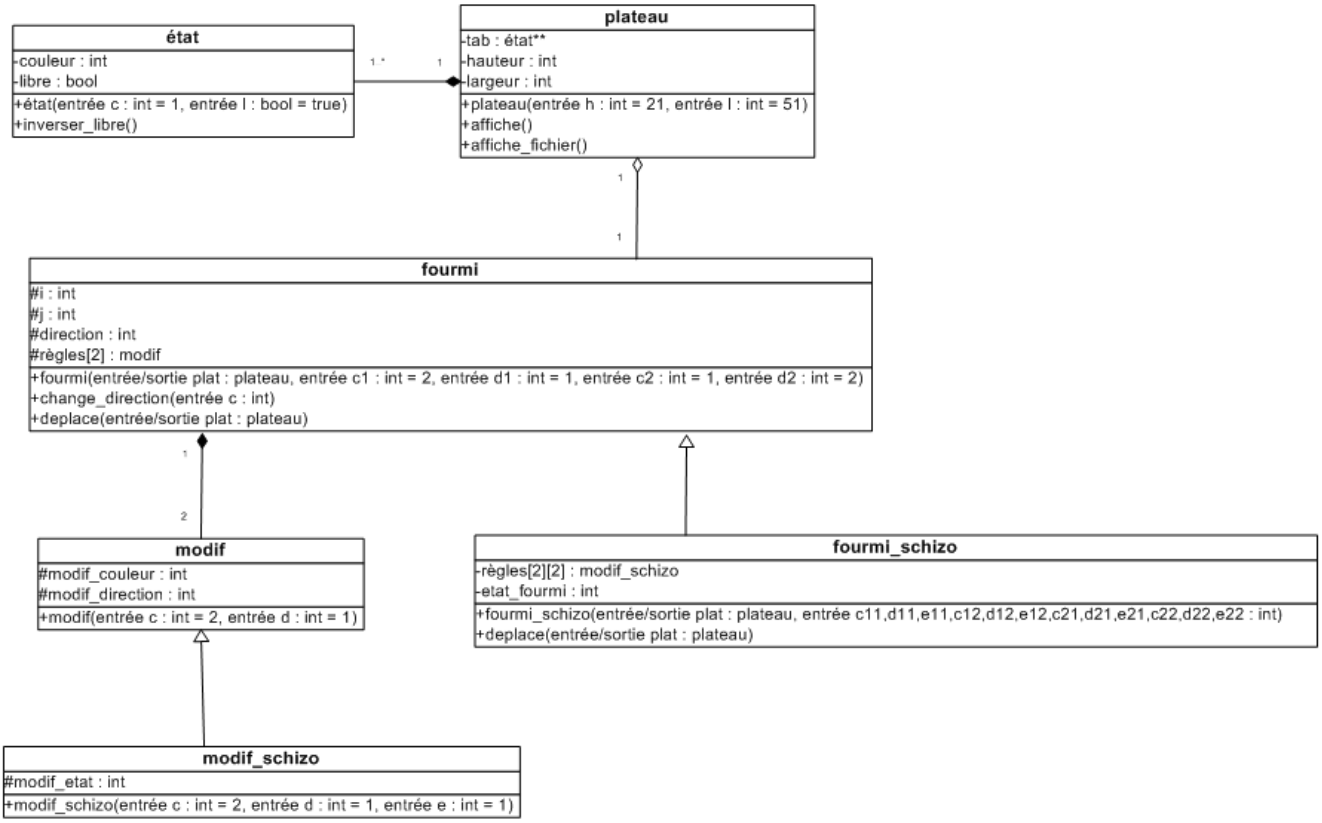
4 PROGRAMMATION DES AUTOMATES CELLULAIRES DU TYPE TURMITE

Il vous est fortement recommandé de lire l'intégralité de cette partie avant de commencer à coder. Les travaux demandés sont mis en évidence avec une couleur bleue.

4.1 DIAGRAMME DE CLASSES

On commence par donner un diagramme de classes qui sera implémenté lors des deux prochaines sous-parties.

Remarque importante : dans le diagramme suivant et dans la suite de l'énoncé on ne fait figurer aucun « getter » ni « setter ». Cela sera à vous de les ajouter quand le besoin s'en fait sentir. En faisant en sorte pour les « setter » qu'ils respectent les mêmes éventuelles contraintes sur les attributs que les constructeurs.



4.2 UN NOUVEAU TYPE DE FOURMI

Déclarer dans un « .h » et implémenter dans un « .cpp » les classes « modif_schizo » et « fourmi_schizo ».

Quelques précisions sur la classe « modif_schizo » :

- L'attribut « modif_etat » vaut 1 ou 2 et indique simplement quel sera le nouvel état de la fourmi après son déplacement.

Quelques précisions sur la classe « fourmi_schizo » :

- « règles » est un tableau de 2 lignes, chacune contenant deux objets de la classe « modif_schizo ». La première ligne concerne le premier état et la deuxième ligne le second état du turmite. Pour chaque ligne le premier objet indique les changements lorsque l'on est sur une case blanche et le second sur une case noire.
  Par exemple pour le turmite dont le tableau est donnée dans la partie 2.2, les attributs du premier objet de la première ligne valent 2, 1 et 2 (dans l'ordre modif_couleur, modif_direction, modif_etat), ceux du deuxième objet de la première ligne valent 1, 1 et 2, ceux du premier objet de la deuxième ligne valent 2, 2 et 2, et ceux du deuxième objet de

la deuxième ligne valent 1, 1 et 1.
On utilisera d'ailleurs ces valeurs comme valeurs pas défaut du constructeur de la classe fourmi, ce qui ne figure pas sur le diagramme de classes précédent.

- La méthode « déplace » réalise le déplacement d'un turmite en suivant le processus en 4 étapes décrit dans la partie « Généralités ».

4.3 UN PROGRAMME OU L'ON FAIT BOUGER NOTRE TURMITE

Mêmes consignes qu'à la partie 3.4. mais cette fois ci avec un turmite.

5 QUELQUES AJOUTS A NOS CLASSES

Rajouter les fonctionnalités suivantes dans les classes précédentes :

- **Donner la possibilité à l'utilisateur de préciser lors de la construction du tableau quelles cases doivent être initialement noires.**
- **Faire des statistiques sur les directions prises par la fourmi. Typiquement le pourcentage de fois où elle aura été dans chacune des quatre directions.**
- **Créer une fourmi plus mobile qui elle se déplacera de deux cases à la fois.**

Bonus :

- **Essayer de faire cohabiter deux (ou plus) fourmis sur le plateau. On définira alors des règles de collisions.**

6 BAREME INDICATIF

Ce barème peut-être amener à évoluer, il n'est donc qu'i**ndicatif**.

- Soutenance : 20 points (détails à venir)
- Partie 3.2. : 10
- Partie 3.3. : 12
- Partie 3.4 : 2
- Partie 4.2. : 10
- Partie 5 : 6
- Bonus : 6

Ce qui fait un total de 60 points, ramené sur 20 par proportionnalité.