Institut National Des Sciences
Appliquées
ROUEN NORMANDIE

Rapport de projet

Projet Integratif Smart Robot

Étudiants :
Yohann Delaplace
Louis Lenoble
Maël Planchot
Dylan Sanson
Nathan Sourdrille

Professeurs référents :
M^{me} Laghmara
M^{r} Delestre

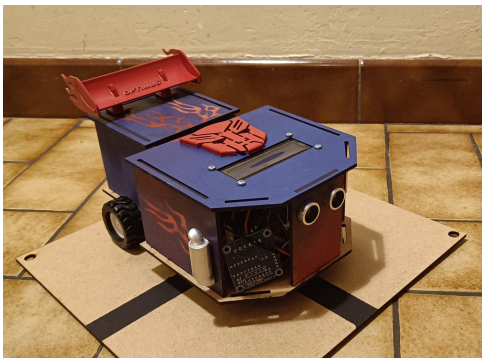


Table des matières

Chapitre 1

Remerciements

Tout d'abord, nous tenons particulièrement à remercier les professeurs référents, M^{me} **Laghmara**, M^{me} **Planterose** ainsi que M^{r} **Delestre** pour leur accompagnement et leur investissement tout au long du projet. Nous tenons également à remercier l'ensemble des intervenants qui nous ont permis de poursuivre notre développement du robot et au département ITI pour tout le matériel mis à notre disposition.

Chapitre 2

Introduction

Dans le cadre de nos études en troisième année à l'INSA Rouen Normandie au sein du département ITI, nous avons reçu l'opportunité de participer au projet Integratif Smart Robot. Celui-ci vise à renforcer et développer des compétences que nous avons abordé en cours. Notre objectif était de concevoir un robot capable de sortir le plus rapidement possible d'un Labyrinthe de forme carré, en suivant une ligne pour se guider.

2.1 Différentes compétences

Ce projet se compose notamment de deux grandes parties, celles-ci nécessitant des compétences en algorithmique et en électronique. Nous nous sommes appuyés sur les outils suivants pour réaliser ce projet :

— `Doxygen` pour la génération de la documentation du code.
— `Fritzing` pour la conception du circuit électronique.
— `GitLab` pour le partage du code source et la création de tickets des actions et demandes.
— `Make` pour la compilation du code source.
— `LaTeX` pour générer un rapport de projet en .pdf après compilation.

2.1.1 Algorithmique

Tout d'abord, la partie algorithmique permet de créer un algorithme et de l'implémenter permettant de trouver le chemin le plus court pour sortir du labyrinthe. Cette implémentation est le résultat de différentes étapes du cycle en V, celles-ci sont détaillés dans la suite du rapport.

2.1.2 Électronique

La partie électronique a pour but de concevoir le robot qui va utiliser l'algorithme pour sortir du labyrinthe. Ainsi, cela nous a poussé à la réflexion du placement des différents composants sur le robot. Cette conception a fait

appel à différentes compétences, comme la création d'un schéma électronique et son câblage associé, le codage sur carte Raspberry, la soudure ...

2.2 Organisation du travail

L'outil GitLab permet le partage du code dans le groupe mais propose également la création de tickets pour suivre les modifications. Ainsi, nous avons créé les tickets suivants afin d'effectuer les tâches requises par le projet. Chaque ticket sera référencé par un # suivit de son numéro (le numéro suit l'ordre de création des tickets) :

— #2 : ce qui concerne latex.
— #3 : ce qui concerne les TADs.
— #4 : ce qui concerne l'analyse descendante (partie algorithmique/ électronique).
— #5 : ce qui concerne le schéma électrique et le fritzing.
— #6 : ce qui concerne la conception préliminaire et la conception détaillée.
— #7 : ce qui concerne la fonction creerLabyrinthe.c.
— #8 : ce qui concerne la fonction analyseFichier.c.
— #9 : ce qui concerne la fonction resolutionLabyrinthe.c.
— #10 : ce qui concerne le code du robot.
— #11 : ce qui concerne tous les types.c.
— #12 : ce qui concerne les tests unitaires.
— #13 : ce qui concerne les makefiles.
— #14 : ce qui concerne l'espace de nommage.
— #16 : ce qui concerne la documentation de la partie électronique.
— #17 : ce qui concerne les tests sur le robot.
— #18 : ce qui concerne le script bash.

Les tickets étaient ainsi à disposition de chaque membre du groupe afin d'assurer la bonne organisation du projet et le suivi des tâches. Cependant, pour assurer le bon déroulement du projet, la répartition du travail était indiqué par le chef de projet. Ainsi, chaque membre du groupe s'est vu attribuer différentes tâches tout au long du projet. En cela, chaque membre du groupe a pu travailler sur différents aspects du projet :

— **Delaplace Yohann**
développement des makefile, bash et tests unitaires,
rédaction du rapport latex.
— **Lenoble Louis (chef de projet)**
organisation du projet,
montage du robot,
développement de la partie algorithmique.

— **Planchot Maël**
développement de la partie électronique et Doxygen.
— **Sanson Dylan**
développement de la partie algorithimique et électronique.
— **Sourdrille Nathan**
développement de la partie algorithmique et électronique,
rédaction du rapport latex et bilan de chaque séance.

Planning :

Afin de maintenir un bon rythme de travail et de pouvoir mener le projet à son terme, nous avons mis en place un planning qui respecte les critères suivants :

— Les tâches à faire avant la séance
— Les tâches à faire durant la séance
— Le déroulement de la séance
— Le bilan de fin de séance

Voici, par exemple, le planning concernant les séances d'électronique :

PLANNING ELEC (EVOLUTIF)			
n° séance	1		2
Tâche(s)	**A faire pendant la séance :**		**A faire pendant la séance:**
	début montage du robot		Louis : Push le fritzing + Faire les branchements
	faire le schéma du câblage		Dylan / Nathan : Faire un maximum de code de l'analyse descendante
	Mise en place du projet gitlab + Spcéification du TAD labyrinthe		Mael / Yohan : Peidnre et percer la maquette
Travail a faire avant la séance	**A faire avant la séance :**		**A faire avant la séance :**
	//		Louis : fritzing
			Tout le monde : analyse descendante
Deroulement de laséance	**Deroulement de la séance:**		**Deroulement de la séance:**
	Dylan/ Nathan : Trouver la maniere de résoudre le probleme		Louis : branchements/ fritzing
	Mael / Yohan : commencer à construire le robot (chassis)		Dylan et nathan : début code et vérification branchements
	louis : réaliser le chema electrique		Louis / yohan / mael : discussion sur l'avancement du percage / de la peinture ...
Bilan fin de séance	**Fin de séance:**		**Fin de séance:**
	Chassis en cours : probleme d'écartement des capteurs		voir si les branchements sont pret à etre installé dans la maquette
	Codes utiles trouvés		voir l'avencement du code et attribuer les fonctions restantes à faire pour la prochaine sceance
	Schéma electrique en cours		

FIGURE 2.1 – planning séances 1 et 2

3		4
A faire pendant la séance:		**A faire pendant la séance:**
Louis /Yohann : mettre composants dans chassis / Finir les branchement moteurs		Développement des .h, des .c vide et des tests unitaires
Nathan / Dylan / Mael : Checker si les focntions et procedures sont utilisables (noms corrects, varibales utilisées ...)		Mise en ordre du latex
Toute l'equipe : Commencer les tests		Compréhension de l'analyse descendante par tout le monde
A faire avant la séance :		**A faire avant la séance:**
Finir les fonctions / procedures		Louis : CP + CD Trouver Plus court chemin / Ecrire le TAD Noeud dans le latex TAD
arret d'urgence, signal sonor, afficher lcd, liste instruction e main		Dylan/Nathan : CP + CD ConversionNoeudsEnOrdres
Mettre sur moodle le fritzing et l'emploi du temps		Mael : CP + CD AnalyserFichier / Macros
		Yohann : CP + CD CreerPassage
Deroulement de la séance:		**Deroulement de la séance:**
Tout le monde : Bilan sur l'avancement		Mise en ordre du latex : syntaxe, liens, corrections de petites erreures ...
Nathan/ Dylan: poursuite du code		Regarder l'analyse descendente tous ensemble pour vérifier que tout le monde comprend
louis : faire le lien des moteurs		Discuter du TAD noeuds et faire les modifications necessaires
yohann / mael : faire les soudures		Regarder les algo, types et TAD inutiles et les retirer
yohann / mael / louis : mettre les composants dans le chassis		
Fin de séance:		**Fin de séance:**
Voir si le code est correct et dispersé le code à corriger		Finir les tests d'implémentation
Voir si les branchments sont terminer et donner le robot à quelqu'un pour qu'il les finissent		

FIGURE 2.2 – planning séances 3 et 4

Matrice des tâches réalisées

Afin de mieux visualiser l'implication de chaque membre du groupe dans les différentes étapes du projet, ci-dessous est présenté la matrice des tâches réalisées (le niveau d'implication dans chaque tâche est indiqué via un pourcentage) :

Tâches	Delaplace Yohann	Lenoble Louis	Planchot Maël	Sanson Dylan	Sourdrille Nathan
schéma électronique fritzing	0	100	0	0	0
choix du placement des capteurs	20	20	20	20	20
création des TAD	20	20	20	20	20
analyse descedante	20	20	20	20	20
recherche algorithme de résolution	0	33	33	33	0
conception détaillée des fonctions principales	0	33	33	33	0
développement des .h (algo)	0	65	0	25	10
développement des .h (élec)	0	0	50	50	0
développement des .c (types)	0	50	0	20	30
développement des .c (algo)	20	40	0	40	0
développement des .c (élec)	0	0	60	30	10
tests unitaires (algo)	90	0	0	5	5
tests unitaires (élec)	20	20	20	20	20
tests d'intégration (algo)	0	50	0	50	0
tests d'intégration (élec)	20	10	50	10	10
makefile	90	0	10	0	0
bash	100	0	0	0	0
mise en place de doxygen	0	0	100	0	0
documentation	0	33	33	33	0
mise en place du latex	50	0	50	0	0
rapport latex	25	5	0	0	70

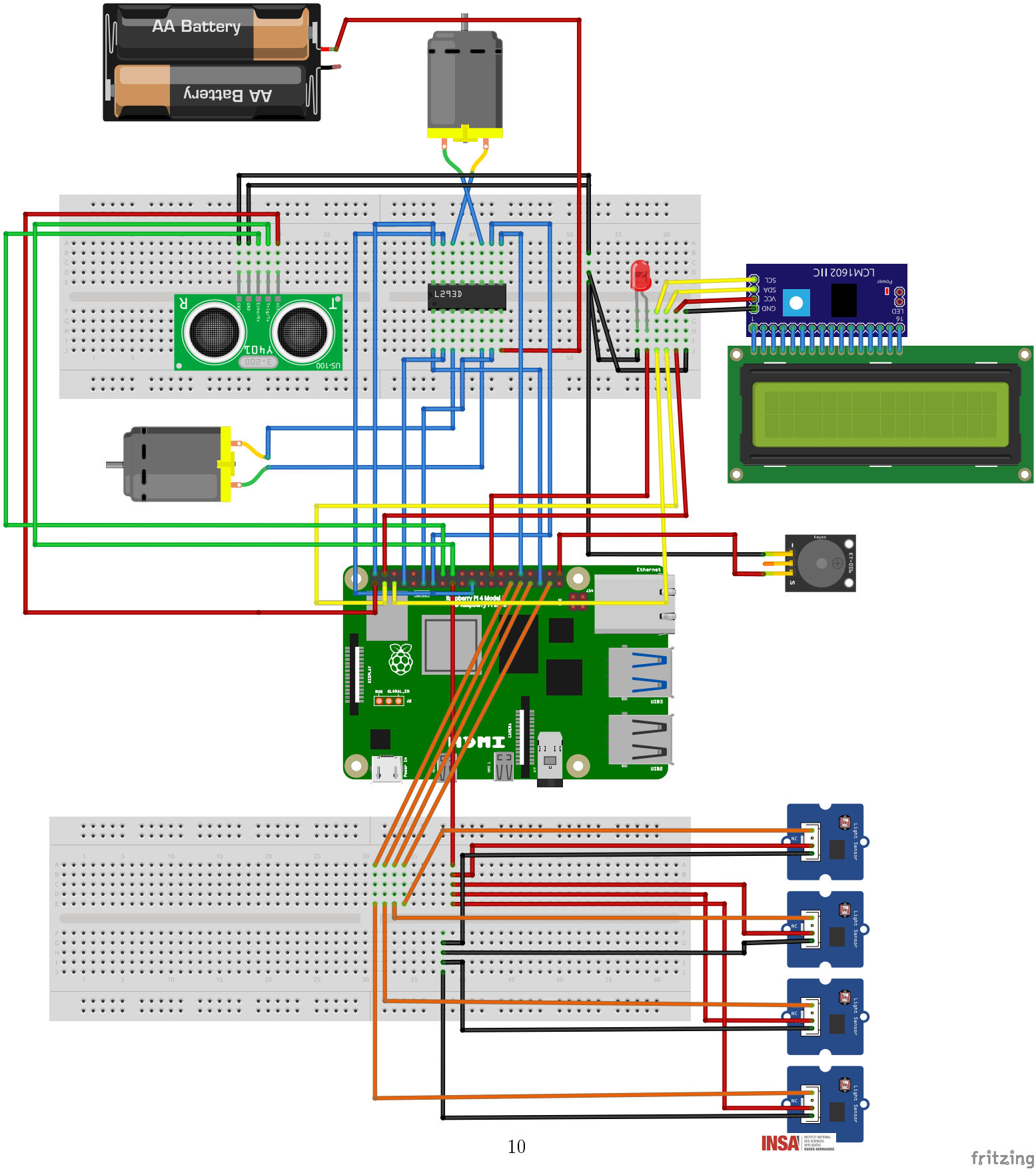
Chapitre 3

Partie électronique : schéma électrique

La toute première étape du projet a été de choisir les composants que nous allions utiliser, leur emplacement sur le robot ainsi que leur branchement. Bien que ce soit une étape qui peu paraître plutôt simple, celle-ci joue un rôle clé dans la préparation du projet. En effet, il faut réussir à s'accorder sur la manière dont va fonctionner le robot pour que chacun puisse travailler de manière autonome. De plus, elle permet d'anticiper l'élaboration de l'analyse descendante.

Ci-dessous sont présentés le schéma électronique mis en place, la liste des composants, ainsi que le bilan énergétique de ceux-là.

Schéma électronique du robot


AA Battery
AA Battery
L293D
LCM1602 IIC
Ethernet
USB3
USB2
Light Sensor

3.1 Explication du fonctionnement des différents composants

3.1.1 Liste des différents composants utilisés

La carte Raspberry est reliée à différents composants ci dessous :

— • **1 capteur à ultrason** (situé sur le devant du robot) permet de détecter les potentiels obstacles se plaçant devant le robot.
— • **4 capteurs suiveurs de ligne** permettent au robot de suivre la ligne noir en la détectant continuellement. Concernant le placement des capteurs, il y en a 3 à l'avant (1 au centre, 1 à gauche, 1 à droite) et 1 à l'arrière (centré avec le capteur placé à l'avant au centre).
— • **2 moteurs** (situés à l'arrière) permettent au robot de se déplacer.
— • **1 carte moteur** permet le contrôle des moteurs.
— • **1 écran LCD** (situé sur le dessus du robot) permet l'affichage des commandes du robot en cours d'exécution.
— • **1 buzzer** (situé à l'arrière du robot) permet d'émettre un son lors de l'arrêt d'urgence.
— • **1 batterie** permet d'alimenter en électricité la carte Raspberry.
— • **2 piles** permettent d'alimenter les moteurs en électricité.
— • **1 bouton** (situé à l'arrière du robot) permet d'ouvrir ou fermer le circuit électrique et ainsi démarrer les moteurs.
— • **1 LED** (situé à l'arrière du robot) permet d'indiquer l'état de charge de la batterie.

3.1.2 Capteur à ultrason

Le capteur à ultrason envoie un signal sonore à l'aide d'un émetteur, puis le signal émis va percuter un obstacle et ainsi être réfléchi. Un capteur a ultrason faisant partie du dispositif va permettre de récupérer le signal réfléchi. Ainsi pour évaluer la distance entre l'objet et le capteur il suffit d'appliquer la formule suivante :

$$\frac{360 \times temps \times V_{son}}{2} = distance$$

Le temps (en seconde) et la distance (en mètre) et V_{son} la vitesse du son dans l'air (344m/s), permettent de déterminer à quelle distance se situe l'objet. Il ne reste ensuite plus qu'à programmer à partir de quelle distance minimale le robot doit s'arrêter. La distance minimale retenue est de 15cm.

3.1.3 Capteurs de ligne

Les capteurs de ligne permettent de détecter une ligne sombre sur un fond clair et inversement. Les capteurs sont calibrés grâce à un potentiomètre situé

sur le capteur. Ces capteurs possèdent 2 modes de sorties :

— Un mode de sortie analogique : plus la ligne est foncée, plus la valeur renvoyée est grande.
— Un mode de sortie tout ou rien : si la couleur du fond est supérieure au seuil défini par le potentiomètre alors le capteur renvoie 1, sinon il renvoie 0.

Dans notre cas, nous avons utilisé le mode tout ou rien pour détecter une ligne.

3.1.4 Bouton on/off

Le bouton on/off (activation manuelle) est branché en série sur le circuit, il sert d'interrupteur afin de ne pas alimenter les moteurs lorsque le robot n'est pas en fonctionnement. Cela permet d'économiser de la batterie.

3.1.5 LED

La LED s'allume quand la valeur du courant qui circule à l'interieur est supérieure à un certain seuil défini en fonction de la couleur de la LED. Dans notre cas la LED est rouge donc elle nécessite une tension comprise entre 1.8V et 2.1V pour s'allumer. Au dessus de cette limite, la led cesse de fonctionner et devient inutilisable. En dessous de cette limite, la LED ne s'allume pas (ou très peu) car elle agit comme une diode.

3.1.6 Carte moteur

La carte moteur permet de contrôler les moteurs. Elle prend en entrée une alimentation externe et une tension définit par la formule suivante :

$$\frac{1024 \times vitesseMoteur}{100}$$

La vitesse du moteur est en pourcentage (de 0 à 100) de la vitesse maximale du moteur. La carte moteur renvoie en sortie une tension et un courant adapté pour le moteur selon ses entrées. Ainsi grâce à cette formule, nous sommes capables de définir la vitesse de rotation du moteur, mais aussi d'activer ou de désactiver un moteur afin de réaliser un virage par exemple.

3.1.7 Buzzer

Le buzzer émet un signal sonore lorsqu'il reçoit une tension en entrée de 3.3V.

3.1.8 Ecran LCD

L'écran LCD possède différents pixels qu'il faut mettre à jour afin d'afficher le contenu voulu. Pour le mettre à jour, on utilise les ports SDA et SCL de l'écran pour communiquer avec la carte Raspberry. Dans notre cas, l'écran LCD affichera les ordres en cours et la détection d'intersection.

3.1.9 La carte Raspberry

C'est la carte électronique principale du robot ayant un OS intégré (linux). Elle est capable d'éxécuter des lignes de codes grâce à une barette de RAM et un processeur intégré. Elle possède en entrée des GPIOs qui permettent de contrôler des éléments externes comme le buzzer ou la carte moteur. Elle est alimentée via un câble USB-C et possède des ports pour brancher un écran, un clavier, une souris ... Une carte réseau est également intégrée, ce qui permet de contrôler la carte à distance via le protocole SSH. Le stockage est assérée via une carte micro-SD.

Ci-dessous une vision simple du robot, pour comprendre sa structure globale

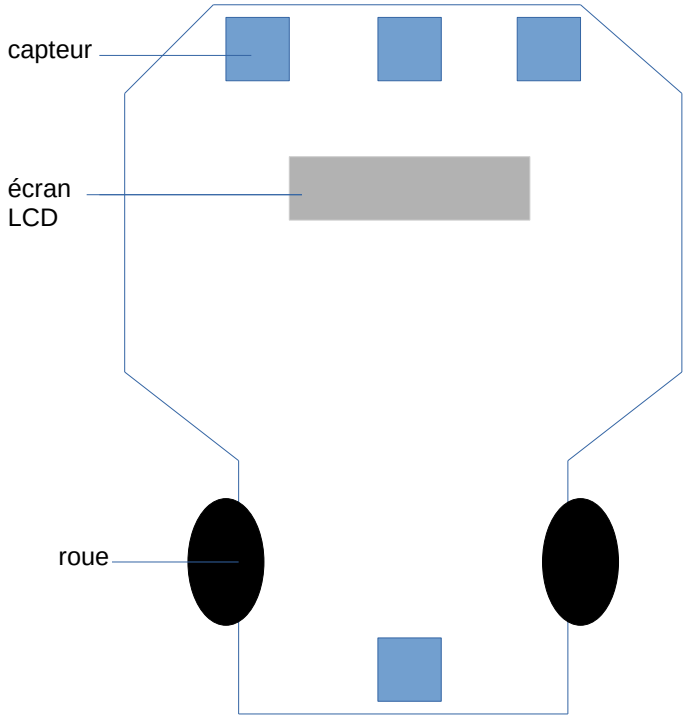


FIGURE 3.1 – Schéma simplifié du robot

3.2 Explication du câblage

La batterie est reliée à la carte Raspberry pour l'alimenter en électricité, ce qui permet à celle-ci d'exécuter le programme permettant le déplacement du robot. De plus, la carte Raspberry est reliée, via les GPIOs, aux différents composants afin de recevoir ou d'émettre des signaux.

Il y a 2 circuits éléctroniques dans le robot :

— le circuit batterie, carte Raspberry et composants.
— le circuit piles, carte moteur et moteurs.

3.2.1 Explication des différents GPIO utilisés

La carte moteur

Dans cette partie, le numéro du GPIO sera celui indiqué sur la carte Raspberry. Le second numéro correspond à celui indiqué sur le composant en question.

— • **GPIO n°19** relié à **EN1** pour indiquer ou non l'activation du moteur 1.
— • **GPIO n°12** relié à **EN2** pour indiquer ou non l'activation du moteur 2.
— • **GPIO n°39** relié à **GRD** pour avoir une masse commune.
— • **GPIO n°2** relié à **VCC** pour alimenter la carte en énergie (tension de +5V).
— • **GPIO n°17 - n°4** (17 pour avancer, 4 pour reculer) pour définir le sens de rotation du moteur 1.
— • **GPIO n°27 - n°9** (27 pour avancer, 9 pour reculer) pour définir le sens de rotation du moteur 2.

L'écran LCD

— • **GPIO n°3** relié à **SDA** pour la communication entre les 2 éléments (I2C).
— • **GPIO n°5** relié à **SCL** pour la communication entre les 2 éléments (I2C).
— • **GPIO n°39** relié à **GND** pour avoir une masse commune.
— • **GPIO n°2** relié à **VCC** pour alimenter en électricité l'écran et la carte LCD.

Le capteur à Ultrason

— • **GPIO n°39** relié à **GND** pour avoir une masse commune.
— • **GPIO n°2** relié à **VCC** pour alimenter la carte en électricité (+3.3V).
— • **GPIO n°24** relié à **Trig/Tx** pour .
— • **GPIO n°23** relié à **Echo/Rx** pour .
— • **GPIO n°39** relié à **GND**.

Les capteurs de lignes

— • **GPIO n°5-n°13-n°26-n°6** (capteur Avant, capteur Droite, capteur Gauche, capteur Arrière) relié à **GND**.
— • **GPIO n°39** relié à **GND**.
— • **GPIO n°2** relié à **VCC**.
— • **GPIO n°20** relié à +.
— • **GPIO n°39** relié à **-**.

3.2.2 Explication du branchement de la LED, du bouton, de l'écran LCD

La **LED** est branchée au circuit des piles et des moteurs, en parallèle. C'est une LED avec résistance directement intégrée, sans danger pour le circuit électronique. **Un bouton** (on/off) a été rajouté entre les piles et le moteur pour permettre la désactivation des moteurs et éviter le déchargement des piles lorsque le robot n'est pas en fonctionnement. **L'écran LCD** est relié à la carte LCD qui permet de la contrôler. **Les moteurs** sont reliées à la carte moteur.

3.2.3 Réalisation du bilan énergétique du robot

Selon la documentation de la carte Raspberry, nous savons que chaque GPIO peut délivrer jusqu'à **3mA** et **3.3V**. Au total on ne doit pas dépasser **120 mA** sur l'ensemble des GPIO. On a donc une puissance max de sortie côté GPIO de **396mW**.
Détail de la consommation énergétique en prenant le pire des cas : chaque capteur est alimenté et transmet des données (carte moteur comprise même si les moteurs ne sont pas en fonctionnement). On utilise la formule
$\mathbf{P = U \times I}$ afin de déterminer la puissance totale de chaque composant

Sur le circuit Raspberry – composants

- • **Les capteurs** ont besoin de **3.3 volts et 2.1 mA** pour fonctionner soit $\mathbf{P = 2{,}1 \times 3.3 \times 4 = 27{,}72\ mW}$
- • **La carte moteur** utilise pour fonctionner une puissance de $\mathbf{P = 5 \times 16 = 80mW}$ (on néglige la liaison sur les pins PWM qui fonctionnent en micro ampère donc ont une puissance négligeable)
- • **Le buzzer** a besoin de **3.3 volts et 10 mA** pour fonctionner soit $\mathbf{P = 33\ mW}$
- • **L'écran LCD** a besoin de **5 volts et 1.1 mA** pour fonctionner soit $\mathbf{P = 5.5\ mW}$
- • **Le capteur à ultrason** a besoin de **5 volts et 2 mA** pour fonctionner soit $\mathbf{P = 10\ mW}$

On a donc une puissance totale de $\mathbf{P = 27{,}72 + 80 + 33 + 5.5 + 10 = 156.22mW}$ au niveau des pins GPIO. Cette puissance étant inférieure au seuil de la carte Raspberry, les composants fonctionneront normalement.

Sur le circuit piles – moteur

- • **La carte moteur** utilise pour fonctionner une puissance de $\mathbf{P = 1.6\ W}$
- • **La LED avec résistance intégrée** utilise une puissance de $\mathbf{P = 10\ mW}$ cette puissance bien inférieure aux autres puissances sera né-

gligée (le constructeur sur le site en annexe n'a pas fourni d'informations supplémentaires à ce sujet)

— • **Les moteurs** sont alimentés par **7.4 volts (3.7 × 2) et 150 mA** donc utilisent une puissance de $\mathbf{P = 3.6W}$

On a donc une puissance totale de $\mathbf{P = 3.6 + 1.6 = 5.2W}$ au niveau des piles pour alimenter le moteur.

Chapitre 4

Principe de fonctionnement du robot

4.1 Fonctionnalités du robot

Dans ce chapitre, nous vous partageons le premier principe de fonctionnement du robot qui a été retenu par l'équipe. Rappelons d'abords les différentes capacités à intégrer au robot :

— fonctionner de manière autonome : le robot doit fonctionner de manière autonome, c'est à dire qu'une fois le script bash lancé, il exécute tous les codes implémentés.

— savoir suivre une ligne et détecter des intersections : le robot dispose de capteurs capablent de détecter une ligne. Le robot doit pouvoir suivre ces lignes s'il veut pouvoir se déplacer dans le labyrinthe. Il doit être capable de distinguer plusieurs lignes et de suivre la bonne (selon l'algorithme explicité plus bas).

— résoudre le labyrinthe : le robot utilise l'algorithme de résolution du labyrinthe pour sortir de celui-ci.

— aller vite : les choix effectués lors du placement des capteurs et de l'implémentation du code du robot ont une influence sur la vitesse du robot dans le parcours du labyrinthe. L'objectif est d'être le groupe le plus rapide à sortir du labyrinthe.

— pouvoir effectuer un arrêt d'urgence : lorsqu'il rencontre un obstacle placé devant lui, le robot doit pouvoir s'arrêter et ne pas heurter cet obstacle.

— émettre un signal sonore : le robot, lorsqu'il s'arrête de manière urgente, émet un signal sonore.

— afficher ses déplacements : le robot suit des ordres de déplacement, qu'il doit pouvoir afficher de manière visible aux personnes extérieurs.

A l'aube du projet, nous avons organisé une réflexion commune sur la

méthode optimale permettant au robot de sortir du labyrinthe le plus vite possible. Dès lors, la réflexion s'est portée sur les composants à utiliser, c'est-à-dire que nous nous sommes demandés comment exploiter au mieux les composants à notre disposition. De plus, nous avons échangé sur la manière dont le robot va parcourir le labyrinthe.

4.2 Principe de parcours : lignes droites et virages

4.2.1 Les lignes droites

Le robot ne peut pas avancer de manière parfaitement droite sur la ligne, il va connaître des écarts de trajectoire qui vont faire que le capteur central ne va plus détecter la ligne. Si cela arrive, il est nécessaire de recentrer le robot sur la ligne. En cela, dès qu'un des deux capteurs latéraux détecte la ligne, le robot se recentre. L'objectif étant qu'il ne diverge pas n'importe où et surtout qu'il arrive dans une intersection (respectivement un virage) le plus droit possible afin de détecter l'intersection (respectivement le virage) suivant efficacement.

4.2.2 Les virages

Le robot a pour objectif de sortir du labyrinthe le plus rapidement possible. En cela, nous avons opté pour une prise de virage dans laquelle le robot reste en mouvement lors de la détection du virage/ de l'intersection.
Prenons l'exemple d'un virage à gauche :
Le capteur gauche détecte une ligne. Le robot continue d'avancer tout en tournant vers la gauche (il a une trajectoire de courbe). Dès lors, le capteur arrière va perdre le contact d'une ligne. Lorsque le capteur arrière détecte une nouvelle ligne, c'est qu'il s'agit de la ligne détectée par le capteur gauche. Alors, le robot s'arrête et effectue une rotation vers la gauche jusqu'à ce que le capteur avant central détecte cette ligne. Cela signifie alors que le robot s'est remis droit.
L'avantage de cette méthode est que le robot continue d'avancer lors de la détection d'une ligne donc lorsqu'il va s'arrêter pour pivoter, la rotation sera inférieure à 45° donc plus rapide qu'une rotation à 90°, ce qui permet un gain de temps.

Chapitre 5

Partie algorithmique : analyse du problème

5.1 Différentes compétences

Au cours de ce projet, nous avons acquis une série de compétences essentielles en programmation et en gestion de projet. Ces compétences incluent non seulement la création et la gestion de documents, l'utilisation des outils de développement, mais également la gestion collaborative du code et la mise en place de bonnes pratiques de développement.

Chaque membre de l'équipe a appris à utiliser les commandes de base de Git, telles que `git add`, `git commit`, `git push` et `git pull`, ce qui a permis de maintenir un répertoire centralisé et d'éviter les conflits de code. Cela nous a également permis de travailler en parallèle sur des parties distinctes du code, tout en maintenant une communication constante sur l'avancement des tâches et les problèmes rencontrés.

Développement du code en C : Le développement en langage C a été au cœur de notre projet. Nous avons créé et structuré différents fichiers sources `.c` et `.h`, en respectant des principes de modularité et de réutilisation de code. Chaque fonction implémentée a été testée pour répondre aux contraintes liées à la résolution du labyrinthe. Nous avons fait le choix d'utiliser de nombreux types : les listes, les listes chaînées, les ensembles et les piles afin de faciliter ensuite l'implémentation des fonctions essentielles à la résolution du labyrinthe.

Tests et correction : Pour nous assurer de la fiabilité de notre code, nous avons mis en place une série de tests unitaires. Ces tests ont permis de valider le bon fonctionnement des différentes fonctions développées, ainsi que de détecter et corriger d'éventuels bugs.

Respect des conventions de codage et collaboration : Un autre aspect essentiel de notre travail en équipe a été le respect des conventions de codage, notamment en termes d'espace de nommage et de lisibilité du code.

5.2 Types abstraits de données

Afin de résoudre le problème du plus court chemin, il nous a semblé judicieux de créer des types nous permettant de manipuler un labyrinthe le plus simplement possible. Après de nombreuses discussions, le groupe a choisi de représenter les types suivants :

Les TAD Pile, Liste, Ensemble sont les mêmes que ceux explicités en cours. On considère, sans plus de démonstration, les types fondamentaux suivants :
— **Direction** = {H, B, G, D}
— **Ordre** = {AV, TD, TG}

Dans la suite, 2 TAD vont être explicités (le TAD Labyrinthe et le TAD CaseEtDirection), détaillant les opérations possibles sur ces types.

5.2.1 Labyrinthe

Nom: Labyrinthe

Utilise: **NaturelNonNul**, Direction, Ensemble

Opérations:
labyrinthe: **NaturelNonNul** × **NaturelNonNul** × **NaturelNonNul** ⇸ **Labyrinthe**
casserMur: **Labyrinthe** × **NaturelNonNul** × **NaturelNonNul** ⇸ **Labyrinthe**
largeur: **Labyrinthe** → **NaturelNonNul**
caseDEntree: **Labyrinthe** → **NaturelNonNul**
caseDeSortie: **Labyrinthe** → **NaturelNonNul**
directionsPossible: **Labyrinthe** × **NaturelNonNul** → Ensemble<Direction>
caseDestination: **Labyrinthe** × **NaturelNonNul** × Direction ⇸ **NaturelNonNul**
casesAdjacentes: **Labyrinthe** × **NaturelNonNul** ⇸ Ensemble<**NaturelNonNul**>
casesAccessibles: **Labyrinthe** × **NaturelNonNul** ⇸ Ensemble<**NaturelNonNul**>
casesNonAccessibles: **Labyrinthe** × **NaturelNonNul** ⇸ Ensemble<**NaturelNonNul**>
directionEntreeEtSortie: **Labyrinthe** → Direction × Direction

Sémantiques:
labyrinthe: L'opération qui permet de créer un labyrinthe muré de taille n^2.
casserMur: L'opération qui permet de créer un accès d'une case à une autre.

largeur: L'opération qui permet de renvoyer la largeur n d'un labyrinthe.

caseDEntree: L'operation qui permet de renvoyer la case d'entrée du labyrinthe.

caseDeSortie: L'opération qui permet de renvoyer la case de sortie du labyrinthe.

directionsPossible: L'opération qui permet de renvoyer les directions possibles depuis une case.

caseDestination: L'opération qui permet de renvoyer la case d'arrivée après l'emprunt de la direction depuis la case précédente.

casesAdjacentes: L'opération qui permet de renvoyer les cases adjacentes à une case.

casesAccessibles: L'opération qui permet de renvoyer les cases accessibles depuis une case.

casesNonAccessibles: L'opération qui permet de renvoyer les cases non accessibles depuis une case.

directionEntreeEtSortie: L'opération qui permet de renvoyer les directions des portes d'entrée et de sortie du labyrinthe.

Préconditions: labyrinthe: entree $\neq$ sortie,
$entree \in [|1,n|] \cup [|n^2-n,n^2|] \cup \{k \in [|1,n|] \mid k \times n\} \cup \{k \in [|1,n|] \mid k \times n+1\}$
$sortie \in [|1,n|] \cup [|n^2-n,n^2|] \cup \{k \in [|1,n|] \mid k \times n\} \cup \{k \in [|1,n|] \mid k \times n+1\}$

casserMur: c1 $\neq$ c2, $c1 \leq largeur(l)^2$ et $c2 \leq largeur(l)^2$

caseDestination: `estPresent(directionsPossible(l,c),d)`

casesAdjacentes: $c \leq largeur(l)^2$

casesAccessibles: $c \leq largeur(l)^2$

casesNonAccessibles: $c \leq largeur(l)^2$

5.2.2 Case et direction

Nom: caseEtDirection

Utilise: **NaturelNonNul**, Direction

Opérations: creerCaseEtDirection: **NaturelNonNul** $\times$ Direction $\rightarrow$ caseEtDirection

fixerCase: caseEtDirection $\times$ **NaturelNonNul** $\nrightarrow$ caseEtDirection

fixerDirection: caseEtDirection $\times$ Direction $\rightarrow$ caseEtDirection

obtenirCase: caseEtDirection $\rightarrow$ **NaturelNonNul**

obtenirDirection: caseEtDirection $\rightarrow$ Direction

Sémantiques: creerCaseEtDirection: L'opération qui permet de créer un objet de type caseEtDirection.

fixerCase: L'opération qui permet d'attribuer une case à un objet de type caseEtDirection.

fixerDirection: L'opération qui permet d'attribuer une direction à un objet de type caseEtDirection.

obtenirCase: L'opération qui permet de renvoyer la case associée à un objet de type caseEtDirection.

obtenirDirection: L'opération qui permet de renvoyer la direction associée à un objet de type caseEtDirection.

Préconditions: fixerCase: $c \leq largeur(l)^2$

5.3 Analyses descendantes

L'analyse descendante consiste à diviser un problème complexe en sous-problèmes plus simples, et ainsi de suite, jusqu'à obtenir des éléments suffisamment simples pour être résolus directement. Pour ce projet, nous avons mis en place deux analyses descendantes, une pour la partie algorithmique du projet, l'autre pour la partie électronique. Celles-ci sont disponibles plus amplement en annexe.

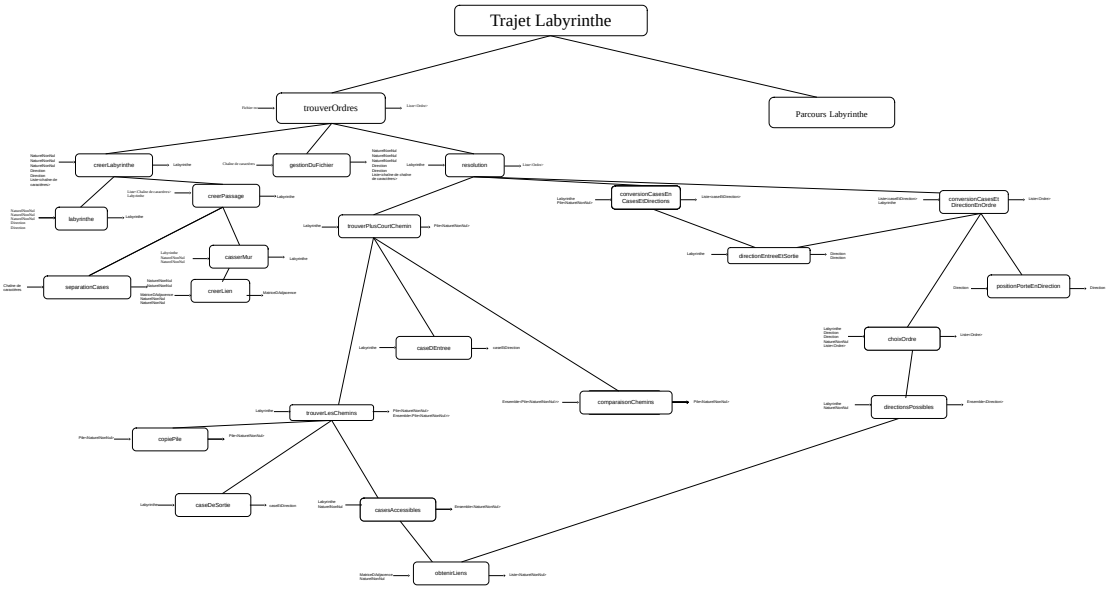
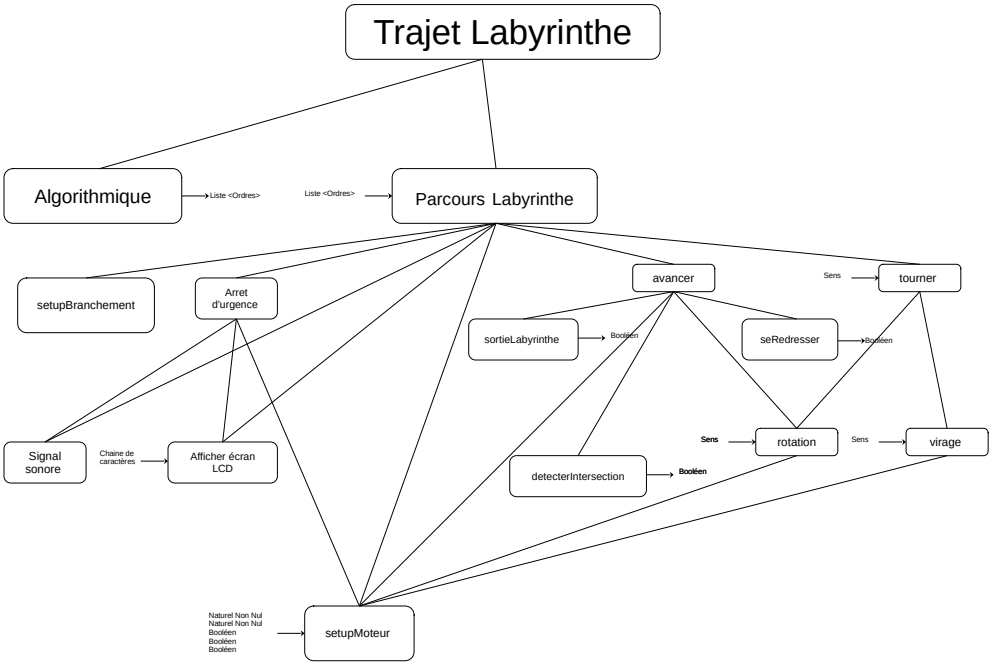


FIGURE 5.1 – Analyse descendante de la partie algorithmique

FIGURE 5.2 – Analyse descendante de la partie électronique



Chapitre 6

Conception préliminaire

Cette partie répertorie les signatures des principaux algorithmes nécessaires à la résolution du problème du labyrinthe.

fonction creerLabyrinthe (caseDEntree : **NaturelNonNul**,
caseDeSortie : **NaturelNonNul**,
largeurLabyrinthe : **NaturelNonNul**,
directionPorteEntree : **Direction**, directionPorteSortie : **Direction**,
liaisonsCases : liste<**ChaineDeCaracteres**>) **: Labyrinthe**

fonction labyrinthe (longueur : **NaturelNonNul**,
caseDeDepart : **NaturelNonNul**,
caseDeFin : **NaturelNonNul**,
direction1 : **Direction**, direction2 : **Direction**) **: Labyrinthe**
Remarque : *Préconditions :*
caseDeDepart différent de caseDeFin : caseDeDepart $\neq$ *caseDeFin*

caseDeDepart appartient à :
$[|1,n|] \cup [|n^2-n,n^2|] \cup \{k \in [|1,n|] \mid k \times n\} \cup \{k \in [|1,n|] \mid k \times n+1\}$

caseDeFin appartient à :
$[|1,n|] \cup [|n^2-n,n^2|] \cup \{k \in [|1,n|] \mid k \times n\} \cup \{k \in [|1,n|] \mid k \times n+1\}$.

procédure creerPassage (**E/S** Laby : **Labyrinthe**,
E liaisonsCases : liste<**ChaineDeCaracteres**>)

procédure separationCases (**E** casesASeparer : **ChaineDeCaracteres**,
S case1 : **NaturelNonNul**, case2 : **NaturelNonNul**)

procédure casserMur (**E/S** Laby : **Labyrinthe**,
E case1 : **NaturelNonNul**, case2 : **NaturelNonNul**)
Remarque : *Préconditions : case1* $\neq$ *case2, case1* $\leq$ *largeur(Laby)*^{2}*, case2* $\leq$ *largeur(Laby)*^{2}.

procédure creerLien (**E/S** matrice : MatriceDAdjacence,
S nn1 : **NaturelNonNul**, nn2 : **NaturelNonNul**)

procédure gestionDuFichier (**E** nomFichier : **ChaineDeCaracteres**, **E/S** listeLiaisons : liste<**ChaineDeCaracteres**, **S** largeur : **NaturelNonNul**, caseEntree : **NaturelNonNul**, caseSortie : **NaturelNonNul**, directionPorteEntree : **Direction**, directionPorteSortie : **Direction**)

fonction resolution (Laby : **Labyrinthe**) **:** liste<ordre>

fonction trouverPlusCourtChemin (Laby : **Labyrinthe**) **:** pile<**NaturelNonNul**>

fonction caseDEntree (Laby : **Labyrinthe**) **:** caseEtDirection

procédure trouverLesChemins (**E** laby : **Labyrinthe**, **E/S** cheminCourant : Pile<**NaturelNonNul**>, cheminsPossibles : Ensemble<Pile<**NaturelNonNul**>)

fonction copiePile (pile : Pile<**NaturelNonNul**>) **:** Pile<**NaturelNonNul**>

fonction caseDeSortie (Laby : **Labyrinthe**) **:** caseEtDirection

fonction casesAccessibles (uneCase : **NaturelNonNul**, Laby : **Labyrinthe**) **:** lesVoisins : Ensemble<**NaturelNonNul**>
Remarque : *Préconditions : uneCase* $\leq$ *largeur(Laby)*^{2}

fonction obtenirLiens (uneCase : **NaturelNonNul**, matrice : MatriceDAdjacence) **:** lesVoisins : Liste<**NaturelNonNul**>

fonction comparaisonChemins (cheminsPossibles : Ensemble<Pile<**NaturelNonNul**>>) **:** Pile<**NaturelNonNul**>

fonction conversionCasesEnCasesEtDirections (Laby : **Labyrinthe**, p : Pile<**NaturelNonNul**>) **:** liste<caseEtDirection

fonction directionEntreeEtSortie (Laby : **Labyrinthe**) **: Direction**, **Direction**

fonction conversionCasesEtDirectionsEnOrdres (Laby : **Labyrinthe**, casesDirections : liste<caseEtDirection) **:** liste<ordres

fonction directionEntreeEtSortie (Laby : **Labyrinthe**) **: Direction**, **Direction**

fonction choixOrdre (Laby : **Labyrinthe**, directionCasePrecedente : **Direction**, directionCaseActuelle : **Direction**, caseActuelle : **NaturelNonNul**, ordres : liste<ordre>) **: Direction**, **Direction**

fonction directionsPossibles (Laby : **Labyrinthe**, nnn : **NaturelNonNul**) **:** Ensemble<**Direction**>

fonction positionPorteEnDirection (d : **Direction**) **: Direction**

procédure afficherListeOrdre (**E** ordres : liste<ordre)

Chapitre 7

Conception détaillée

Le groupe s'est concerté pour se fixer des contraintes de codage afin de faciliter la relecture du code par d'autres membres du groupe (tabulation, positionnement des parenthèses et accolades...). De plus, la conception détaillée nous permet de nous guider pour la phase d'écriture des algorithmes en C.

7.1 Principes de développement

7.1.1 Structure de l'algorithme

Pour implémenter l'algorithme nous avons décidé d'utiliser de nombreux types : listes, listes chaînées, ensembles ... afin d'avoir peu de fonctions à implémenter ensuite pour résoudre le problème du labyrinthe. De plus, l'ensemble de ces types permet une implémentation plus structurée, plus claire et plus évidente de par l'utilisation de fonctions et donc la création d'algorithmes plutôt courts, car de nombreuses opérations sont définies dans les types utilisées.

7.1.2 Les espaces de nommage

Les espaces de nommages sont primordiales lors de l'écriture collaborative d'algorithmes afin de garder un code clair et harmonieux. Alors, nous avons définis les conventions suivantes :

— Chaque type commence par une lettre majuscule, avec chaque nouveau mot également en majuscule. Exemple : `ListeDOrdre`. Chaque type est préfixé par ses initiales. Exemple : `LDO_ListeDOrdre`.

— Les fonctions et procédures sont toujours préfixées par le type qu'elles manipulent. Elles commencent toutes par une minuscule, avec chaque nouveau mot en majuscule. Exemple : `LAB_initialisationLabyrinthe`.

— Les variables sont toujours écrites en minuscules, avec une majuscule pour chaque nouveau mot. Exemple : `positionActuelle`.
— L'ouverture d'une accolade se fait sur la même ligne que l'instruction dont elle dépend, et la fermeture de l'accolade est sur une nouvelle ligne.
— Une indentation est ajoutée à chaque entrée dans une boucle ou une condition.

7.2 Conception détaillée des fonctions principales

Ci-dessous sont répertoriées les principales signatures des algorithmes permettant de résoudre le labyrinthe.

7.2.1 analyse Fichier

fonction analyseFichier (fichier : fichier.txt) **: NaturelNonNul**,**NaturelNonNul**, **NaturelNonNul**,Liste<ChaineDeCaracteres>

Déclaration taille : **NaturelNonNul**, entree : **NaturelNonNul**, sortie : **NaturelNonNul**, passages : Liste<ChaineDeCaracteres>

debut
taille ← LireLigne(fichier)
entree ← LireLigne(fichier)
sortie ← LireLigne(fichier)
tant que nonfichier.estFinDeFichier() **faire**
Liste.inserer(passages,LireLigne(fichier))
fintantque
retourner taille, entree, sortie, passages
fin

7.2.2 Cases est presente dans pile

fonction CasesEstPresenteDansPile (pileCase : Pile<noeud>, pile : Pile<noeud>) **: Booleen**

Déclaration estPresent : **Booleen**, pileCaseTemp : Pile<noeud> , pileTemp : Pile<noeud>

debut
estPresent ← Faux
tant que non estVide(pile) ou (estPresent = Vrai) **faire**
si obtenirElement(pile) = obtenirElement(pileCase) **alors**
pileCaseTemp ← CopiePile(pileCase)
pileTemp ← CopiePile(pile)
tant que non estVide(pileCaseTemp) et (obtenirElement(pileTemp) = obtenirElement(pileCaseTemp)) **faire**

```
            dépiler(pileTemp)
            dépiler(pileCaseTemp)
        fintantque
        si estVide(pileCaseTemp) alors
            estPresent ← Vrai
        finsi
    sinon
        dépiler(pile)
    finsi
  fintantque
  retourner estPresent
fin
```

7.2.3 Comparaison chemins

fonction ComparaisonChemins (lesPiles : Ensemble<Pile<noeud»**) :** Pile<noeud>

```
  Déclaration  PlusCourtCheminDeSortie : Naturel,
               pileDuPlusCourtChemin : Pile<noeud>
debut
  PlusCourtCheminDeSortie ← 1000
  pileDuPlusCourtChemin ← Pile()
  pour pile dans lesPiles faire
  longeurDeLaPile ← longueur(pile)
  si longeurDeLaPile < PlusCourtCheminDeSortie alors
    PlusCourtCheminDeSortie ← longeurDeLaPile
    pileDuPlusCourtChemin ← pile
  finsi
  retourner pileDuPlusCourtChemin
fin
```

7.2.4 conversion cases en cases et directions

fonction conversionCasesEnCasesEtDirections (laby : Labyrinthe, p : Pile<**NaturelNonNul**>)
: Liste<caseEtDirection>

```
  Déclaration  casesDirections : Liste<caseEtDirection>,
               caseActuelle, casePrecedente : NaturelNonNul,
               porteEntree, porteSortie : Direction
debut
  caseActuelle ← obtenirElement(p)

  casesDirections ← listeVide()
```

```
    directionEntreeEtSortie(laby, porteEntree, porteSortie)

    inserer(casesDirections, caseEtDirection(caseActuelle, positionPorteEnDi-
    rection(porteSortie)), 1)

    depiler(p)

    tant que non estVide(p) faire
        casePrecedente ← obtenirElement(p)

        si casePrecedente < caseActuelle alors
            si casePrecedente + 1 = caseActuelle alors
                inserer(casesDirections, caseEtDirection(casePrecedente, D), 1)
            finsi
            inserer(casesDirections, caseEtDirection(casePrecedente, B), 1)
        finsi
        si (casePrecedente - 1) = caseActuelle alors
            inserer(casesDirections, caseEtDirection(casePrecedente, G), 1)
        finsi
        inserer(casesDirections, caseEtDirection(casePrecedente, H), 1)
        depiler(p)

        caseActuelle ← casePrecedente
    fintantque
    retourner casesDirections
fin
```

7.2.5 conversion cases et directions en ordres

```
fonction conversionCasesEtDirectionsEnOrdres (laby : Labyrinthe,
casesDirections : Liste<caseEtDirection>) : Liste<Ordre>
    Déclaration  ordres : Liste<Ordre>, directionCaseActuelle,
                 directionCasePrecedente,
                 porteEntree, porteSortie : Direction,
                 caseActuelle, iLecture, iEcriture : NaturelNonNul
debut
    ordres ← listeVide()
    iEcriture ← 1
    iLecture ← 1
    inserer(ordres, AV, iEcriture)
    directionEntreeEtSortie(laby, porteEntree, porteSortie)
    directionCasePrecedente ← positionPorteEnDirection(porteEntree)
    tant que non estVide(casesDirections) faire
```

```
        directionCaseActuelle ← obtenirDirection(obtenirElement(casesDirections,
        iLecture))
        caseActuelle ← obtenirCase(obtenirElement(casesDirections, iLecture))

        iLecture ← iLecture + 1
        choixOrdre(ordres, iEcriture, laby, directionCasePrecedente,
        directionCaseActuelle, caseActuelle)
        directionCasePrecedente ← directionCaseActuelle
    fintantque
    retourner ordres
fin
```

7.2.6 choix ordre

```
procédure choixOrdre (E/S ordres : Liste<Ordre>, iEcriture :
NaturelNonNul, E l : Labyrinthe, directionCasePrecedente,
directionCaseActuelle : Direction, caseActuelle : NaturelNonNul)
    Déclaration   o : Ordre, directionsPossibles : Ensemble<Direction>,
                  d : Direction, estIntersection : Booleen
debut
    si directionCasePrecedente = directionCaseActuelle alors
        directionsPossibles ← obtenirDirectionsPossibles(l, caseActuelle)

        estIntersection ← Faux

        pour chaque d de directionsPossibles
            cas où directionCaseActuelle vaut
                H:
                    si non(d = H ou d = B) alors
                        estIntersection ← Vrai
                    finsi
                B:
                    si non(d = H ou d = B) alors
                        estIntersection ← Vrai
                    finsi
                G:
                    si non(d = G ou d = D) alors
                        estIntersection ← Vrai
                    finsi
                D:
                    si non(d = G ou d = D) alors
                        estIntersection ← Vrai
                    finsi
```

```
            fincas
        finpour
        si estIntersection alors
            inserer(ordres, AV, iEcriture)

            iEcriture ← iEcriture + 1
        finsi
    sinon
        cas où directionCasePrecedente vaut
            H:
                si directionCaseActuelle = G alors
                    o ← TG
                sinon
                    o ← TD
                finsi
            B:
                si directionCaseActuelle = D alors
                    o ← TG
                sinon
                    o ← TD
                finsi
            G:
                si directionCaseActuelle = B alors
                    o ← TG
                sinon
                    o ← TD
                finsi
            D:
                si directionCaseActuelle = H alors
                    o ← TG
                sinon
                    o ← TD
                finsi
        fincas
        inserer(ordres, o, iEcriture)

        iEcriture ← iEcriture + 1

        inserer(ordres, AV, iEcriture)

        iEcriture ← iEcriture + 1
    finsi
fin
```

7.2.7 Copie pile

fonction CopiePile (pile : Pile<noeud>) **:** Pile<noeud>
debut
  pileCopie ← Pile()
  tant que non estVide(pile) **faire**
    empiler(pileCopie , obtenirElement(pile))
    dépiler(pile)
  fintantque
  retourner pileCopie
fin

7.2.8 creer passage

procédure creerPassage **(E/S** laby : Labyrinthe **E** case1, case2 : **Naturel-NonNul)**
debut
  si EstCaseAccessible(Laby, case1, case2) **alors**
    casserMur(Laby, case1, case2)
  finsi
fin

7.2.9 Longueur

fonction LongueurPile (pile : Pile<noeud>) **: NaturelNonNul**

  Déclaration longueurDeLaPile : **Naturel**

debut
  longueurDeLaPile ← 0
  tant que non estVide(pile) **faire**
    longueurDeLaPile ← longueurDeLaPile + 1
    dépiler(pile)
  fintantque
  retourner longueurDeLaPile
fin

7.2.10 plus court chemin

fonction trouverPlusCourtChemin (labyrinthe : Labyrinthe) **:** Pile<**NaturelNonNul**>

  Déclaration cheminCourant : Pile<**NaturelNonNul**>, premiereCase : **NaturelNonNul**, cheminsPossibles : Ensemble<Pile<**NaturelNonNul**>>, cheminLePlusCourt : Pile<**NaturelNonNul**>

debut
  cheminCourant ← pile()

premiereCase ← caseEntree(labyrinthe)
cheminCourant ← empiler(cheminCourant, premiereCase)
cheminsPossibles ← ensemble()
trouverlesCheminslabyrinthe, cheminCourant, cheminsPossibles
cheminLePlusCourt ← comparaisonChemins(cheminsPossibles)
retourner cheminLePlusCourt
fin

7.2.11 trouver les chemins

procédure trouverLesChemins (**E** labyrinthe : Labyrinthe, **E/S** cheminCourant : Pile<**NaturelNonNul**>, cheminsPossibles : Ensemble<Pile<**NaturelNonNul**> >)

Déclaration uneCase : **NaturelNonNul**

debut
si non(estVide(cheminCourant)) **alors**
si obtenirElement(cheminCourant) = caseSortie(labyrinthe) **alors**
ajouter(cheminsPossibles, cheminCourant)
sinon
pour chaque uneCase **de** casesAccessibles(obtenirElement(cheminCourant))

si non(estPresent(cheminCourant, uneCase)) **alors**
cheminCourant ← empiler(cheminCourant, uneCase)
trouverPlusCourtCheminR(labyrinthe, cheminCourant, cheminsPossibles)
finsi
finpour
finsi
cheminCourant ← depiler(cheminCourant)
finsi
fin

7.3 Développement en langage C

Dès lors que l'ensemble des éléments essentiels à la résolution du problème du labyrinthe ont été identifié, vient l'étape de l'implémentation en langage C. D'abord, il s'agit de créer les interfaces à l'aide des .h . Comme nous utilisons de nombreux types, il est préférable de présenter directement les .h et .c liés à la sortie du labyrinthe.

7.3.1 Les .h principaux

Ci dessous sont présentés les fichiers .h principaux.

```c
/**
 * @file trouverLesOrdres.h
 * @brief Interface pour la recherche d'ordres pour sortir du
    labyrinthe.
 */

#ifndef TROUVERLESORDRES
#define TROUVERLESORDRES

#include <errno.h>
#include <assert.h>
#include <stdio.h>
#include <stdbool.h>
#include "../../include/Labyrinthe/gestionFichier.h"
#include "../../include/Labyrinthe/initialisationLabyrinthe.h"
#include "../../include/Labyrinthe/traitementLabyrinthe.h"
#include "../../include/Types/ListeDOrdre.h"


/**
 * @brief Trouve les ordres pour sortir du labyrinthe.
 *
 * @param nomFichier [E] nom du fichier contenant le labyrinthe
    .
 *
 * @pre
 * - `nomFichier` doit exister.
 */
LDO_ListeDOrdre TLO_trouverOrdres(char* nomFichier);

#endif
```

```c
/**
 * @file traitementLabyrinthe.h
 * @brief Interface pour la résolution des labyrinthes.
 */

#ifndef TRAITEMENTLABYRINTHE
#define TRAITEMENTLABYRINTHE

#include <errno.h>
#include <assert.h>

#include "../../include/Types/labyrinthe.h"
#include "../../include/Types/ListeDOrdre.h"
#include "../../include/Labyrinthe/trouverChemin.h"
#include "../../include/Labyrinthe/casesEnOrdres.h"

/**
 * @brief Résout le labyrinthe donné et retourne la liste des
```

```
     ordres nécessaires pour le parcourir.
 *
 * @param laby [E] Le labyrinthe à résoudre.
 *
 * @return Une liste d'ordres représentant les étapes pour
     traverser le labyrinthe.
 *
 * @pre
 * - Le labyrinthe `laby` doit être correctement initialisé.
 * - Les cases d'entrée et de sortie du labyrinthe doivent être
     valides.
 */
LDO_ListeDOrdre TL_resolution(LAB_Labyrinthe laby);

#endif
```

```
/**
 * @file trouverChemin.h
 * @brief Interface pour la recherche de chemins dans un
     labyrinthe.
 */

#ifndef TROUVERCHEMIN
#define TROUVERCHEMIN

#include <errno.h>
#include <assert.h>
#include <stdbool.h>
#include "../../include/Types/labyrinthe.h"
#include "../../include/Types/PileDeNNN.h"
#include "../../include/Types/EnsembleDePileDeNNN.h"
#include "../../include/Types/caseEtDirection.h"
#include "../../include/Types/EnsembleDeNNN.h"

/**
 * @brief Trouve tous les chemins possibles dans le labyrinthe.
 *
 * @param laby [E] Le labyrinthe à explorer.
 * @param cheminCourant [E/S] La pile représentant le chemin en
     cours d'exploration.
 * @param cheminsPossibles [E/S] L'ensemble des chemins trouvés
     .
 *
 * @pre
 * - `cheminCourant` doit être une pile initialisée.
 * - `cheminsPossibles` doit être un ensemble initialisé.
 */
void TC_trouverLesChemins(LAB_Labyrinthe laby, PDNNN_PileDeNNN
    *cheminCourant, EDPDNNN_EnsembleDePileDeNNN *
    cheminsPossibles);

/**
 * @brief Compare les chemins possibles pour déterminer le plus
     court.
```

```
33  *
34  * @param cheminsPossibles [E] L'ensemble des chemins à
        comparer.
35  *
36  * @return Une pile représentant le chemin le plus court trouvé
        dans l'ensemble.
37  *
38  * @pre
39  * - `cheminsPossibles` doit contenir au moins un chemin valide
        .
40  */
41 PDNNN_PileDeNNN TC_comparaisonChemins(
        EDPDNNN_EnsembleDePileDeNNN cheminsPossibles);
42
43 /**
44  * @brief Trouve le chemin le plus court dans le labyrinthe
        entre l'entrée et la sortie.
45  *
46  * @param laby [E] Le labyrinthe à analyser.
47  *
48  * @return Une pile représentant le chemin le plus court.
49  *
50  * @pre
51  * - Le labyrinthe `laby` doit être correctement initialisé.
52  */
53 PDNNN_PileDeNNN TC_trouverPlusCourtChemin(LAB_Labyrinthe laby);
54
55 #endif
```

```
1  /**
2   * @file casesENOrdres.h
3   * @brief convertis les cases données en ordre pour le robot
4   */
5
6
7  #ifndef CASESENORDRES
8  #define CASESENORDRES
9
10 #include <errno.h>
11 #include <assert.h>
12 #include <stdbool.h>
13 #include "../../include/Types/Ordre.h"
14 #include "../../include/Types/Direction.h"
15 #include "../../include/Types/caseEtDirection.h"
16 #include "../../include/Types/labyrinthe.h"
17 #include "../../include/Types/PileDeNNN.h"
18 #include "../../include/Types/ListeDOrdre.h"
19 #include "../../include/Types/ListeDeCaseEtDirection.h"
20 #include "../../include/Types/EnsembleDeDirection.h"
21
22
23 /**
24  * @brief Détermine l'ordre à choisir à une certaine position
        dans le labyrinthe.
```

```
25  *
26  * @param laby [E] Le labyrinthe à explorer.
27  * @param directionCasePrecedente [E] la direction de la case
        précédente.
28  * @param directionCaseActuelle [E] la direction de la case
        actuelle.
29  * @param caseActuelle [E] la case actuelle.
30  * @param ordres [E/S] la liste des ordres à effectuer.
31  */
32 void CEO_choixOrdre(LAB_Labyrinthe laby, Direction
        directionCasePrecedente, Direction directionCaseActuelle,
        unsigned int caseActuelle, LDO_ListeDOrdre *ordres);
33
34 /**
35  * @brief renvoie la direction en fonction de la position de la
        porte.
36  *
37  * @param la position de la porte de la porte.
38  *
39  * @return la direction.
40  */
41  Direction CEO_positionPorteEnDirection(Direction d);
42
43 /**
44  * @brief convertis les cases données en type caseEtDirection.
45  *
46  * @param une pile de cases à a parcourir.
47  *
48  * @return une liste représentant les cases et les directions à
        suivre.
49  */
50 LDCD_ListeDeCaseEtDirection
        CEO_conversionCasesEnCasesEtDirections(LAB_Labyrinthe laby,
        PDNNN_PileDeNNN p);
51
52 /**
53  * @brief convertis les cases et les directions en ordres.
54  *
55  * @param une liste représentant les cases et les directions à
        suivre.
56  * @param le labyrinthe à explorer.
57  *
58  * @return une liste représentant les ordres à suivre.
59  */
60 LDO_ListeDOrdre CEO_conversionCasesEtDirectionsEnOrdres(
        LDCD_ListeDeCaseEtDirection casesDirections, LAB_Labyrinthe
        laby);
61
62 #endif
```

```
1  /**
2   * @file gestionFichier.h
3   * @brief Interface pour la lecture du fichier contenant le
        labyrinthe.
```

```
 */

#ifndef GESTIONFICHIER
#define GESTIONFICHIER

#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <stdbool.h>
#include <string.h>
#include "../../include/Types/ListeDeCDC.h"
#include "../../include/Types/Direction.h"


/**
 * @brief Lit le fichier pour pouvoir représenter le labyrinte.
 *
 * @param nomFichier [E] Le nom du fichier contenant le
     labyrinthe.
 * @param largeur [E/S] largeur du labyrinthe.
 * @param caseEntree [E/S] case d'entree du labyrinthe.
 * @param caseSortie [E/S] case de sortie du labyrinthe.
 * @param directionPorteEntree [E/S] direction de la porte d'
     entrée du labyrinthe.
 * @param directionPorteSortie [E/S] direction de la porte de
     sortie du labyrinthe.
 * @param listeLiaisons [E/S] liste des liaisons entre les
     cases du labyrinthe.
 *
 * @pre
 * - `nomFichier` doit exister.
 */
void GF_gestionDuFichier(char* nomFichier, unsigned int *
    largeur, unsigned int* caseEntree, unsigned int *caseSortie
    , Direction *directionPorteEntree, Direction *
    directionPorteSortie, LDCDC_ListeDeCDC *listeLiaisons);

#endif
```

7.3.2 Les .c principaux

Ci dessous sont présentés les .c des .h présentés ci-dessus. Veuillez trouver les .c des types en Annexe.

```
#include <errno.h>
#include <assert.h>
#include <stdio.h>
#include <stdbool.h>
#include "../../include/Labyrinthe/gestionFichier.h"
#include "../../include/Labyrinthe/initialisationLabyrinthe.h"
#include "../../include/Labyrinthe/traitementLabyrinthe.h"
#include "../../include/Types/caseEtDirection.h"
```

```c
#include "../../include/Types/ListeDOrdre.h"
#include "../../include/Types/Direction.h"
#include "../../include/Types/ListeDeCDC.h"
#include "../../include/Types/labyrinthe.h"
#include "../../include/Labyrinthe/trouverLesOrdres.h"
#include "../../include/Labyrinthe/trouverChemin.h"
#include "../../include/Types/PileDeNNN.h"
#include "../../include/Labyrinthe/casesEnOrdres.h"
#include "../../include/Labyrinthe/traitementLabyrinthe.h"



void afficherListeOrdre(LDO_ListeDOrdre o){
  printf("La liste est \n");
  unsigned int i = 1;
  while (i<= LDO_longueur(o)) {
    switch (LDO_obtenirElement(o, i)) {
    case AV : printf("AV\n");
      break;
    case TG : printf("TG\n");
      break;
    case TD : printf("TD\n");
      break;
    }
    i++;
  }
  printf(".\n");
}


int main(int argc, char** argv){
  char *nomFichier = argv[1];
  unsigned int largeur, caseEntree, caseSortie;
  Direction directionPorteEntree, directionPorteSortie;
  LDCDC_ListeDeCDC listeLiaisons = LDCDC_liste();
  LAB_Labyrinthe laby;
  LDO_ListeDOrdre ordres;

  GF_gestionDuFichier(nomFichier, &largeur, &caseEntree, &
      caseSortie, &directionPorteEntree, &directionPorteSortie,
       &listeLiaisons);
  laby = IL_creerLabyrinthe(largeur, caseEntree, caseSortie,
      directionPorteEntree, directionPorteSortie, listeLiaisons
      );
  ordres = TL_resolution(laby);
  afficherListeOrdre(ordres);

  /* Partie Robot */

  return 0;
}
```

```c
#include <errno.h>
```

```c
#include <assert.h>
#include <stdio.h>
#include <stdbool.h>
#include "../../include/Labyrinthe/gestionFichier.h"
#include "../../include/Labyrinthe/initialisationLabyrinthe.h"
#include "../../include/Labyrinthe/traitementLabyrinthe.h"
#include "../../include/Types/ListeDOrdre.h"
#include "../../include/Types/Direction.h"
#include "../../include/Types/ListeDeCDC.h"
#include "../../include/Types/labyrinthe.h"
#include "../../include/Labyrinthe/trouverLesOrdres.h"

LDO_ListeDOrdre TLO_trouverOrdres(char* nomFichier){
  unsigned int largeur, caseEntree, caseSortie;
  Direction directionPorteEntree, directionPorteSortie;
  LDCDC_ListeDeCDC listeLiaisons = LDCDC_liste();
  LAB_Labyrinthe laby;
  LDO_ListeDOrdre ordres;

  GF_gestionDuFichier(nomFichier, &largeur, &caseEntree, &
      caseSortie, &directionPorteEntree, &directionPorteSortie,
       &listeLiaisons);
  laby = IL_creerLabyrinthe(largeur, caseEntree, caseSortie,
      directionPorteEntree, directionPorteSortie, listeLiaisons
      );
  ordres = TL_resolution(laby);
  return ordres;
}
```

```c
#include <errno.h>
#include <assert.h>

#include "../../include/Labyrinthe/traitementLabyrinthe.h"
#include "../../include/Types/labyrinthe.h"
#include "../../include/Types/PileDeNNN.h"
#include "../../include/Types/ListeDeCaseEtDirection.h"
#include "../../include/Types/ListeDOrdre.h"
#include "../../include/Labyrinthe/trouverChemin.h"
#include "../../include/Labyrinthe/casesEnOrdres.h"

LDO_ListeDOrdre TL_resolution(LAB_Labyrinthe laby){
  PDNNN_PileDeNNN p = TC_trouverPlusCourtChemin(laby);
  LDCD_ListeDeCaseEtDirection casesDirections =
      CEO_conversionCasesEnCasesEtDirections(laby, p);
  LDO_ListeDOrdre ordres =
      CEO_conversionCasesEtDirectionsEnOrdres(casesDirections,
      laby);
  return ordres;
}
```

```c
#include <stdio.h>
#include <errno.h>
#include <assert.h>
```

```
4  #include <stdbool.h>
5  #include "../../include/Types/labyrinthe.h"
6  #include "../../include/Types/PileDeNNN.h"
7  #include "../../include/Types/EnsembleDePileDeNNN.h"
8  #include "../../include/Types/caseEtDirection.h"
9  #include "../../include/Types/EnsembleDeNNN.h"
10
11
12 void TC_trouverLesChemins(LAB_Labyrinthe laby, PDNNN_PileDeNNN
       *cheminCourant, EDPDNNN_EnsembleDePileDeNNN *
       cheminsPossibles){
13   unsigned int uneCase;
14   EDNNN_EnsembleDeNNN lesCasesAccessibles;
15
16   if (!(PDNNN_estVide(*cheminCourant))){
17     if (PDNNN_obtenirElement(*cheminCourant) == CD_obtenirCase(
           LAB_caseDeSortie(laby))){
18       EDPDNNN_ajouter(cheminsPossibles, PDNNN_copier(*
             cheminCourant));
19     }
20     else{
21       lesCasesAccessibles = LAB_casesAccessibles(laby,
             PDNNN_obtenirElement(*cheminCourant));
22       while (!(EDNNN_cardinalite(lesCasesAccessibles) == 0)){
23         uneCase = EDNNN_obtenirPremierElement(
               lesCasesAccessibles);
24         EDNNN_retirer(&lesCasesAccessibles,uneCase);
25         if (!(PDNNN_estPresent(*cheminCourant,uneCase))){
26           PDNNN_empiler(cheminCourant,uneCase);
27           TC_trouverLesChemins(laby,cheminCourant,
                 cheminsPossibles);
28         }
29       }
30     }
31     PDNNN_depiler(cheminCourant);
32   }
33 }
34
35
36
37 PDNNN_PileDeNNN TC_comparaisonChemins(
       EDPDNNN_EnsembleDePileDeNNN cheminsPossibles){
38   PDNNN_PileDeNNN pileCourante;
39   PDNNN_PileDeNNN plusPetitePile;
40   unsigned int longueurCourante;
41   unsigned int plusPetiteLongueur;
42   EDPDNNN_EnsembleDePileDeNNN lesCheminsPossibles =
         cheminsPossibles;
43   plusPetitePile = EDPDNNN_obtenirPremierElement(
         lesCheminsPossibles);
44   EDPDNNN_retirer(&lesCheminsPossibles,plusPetitePile);
45   plusPetiteLongueur = PDNNN_longueur(plusPetitePile);
46   while (!(EDPDNNN_cardinalite(lesCheminsPossibles) == 0)){
47     pileCourante = EDPDNNN_obtenirPremierElement(
```

```
        lesCheminsPossibles);
48      EDPDNNN_retirer(&lesCheminsPossibles,pileCourante);
49      longueurCourante = PDNNN_longueur(pileCourante);
50      if (longueurCourante < plusPetiteLongueur){
51        plusPetitePile = pileCourante;
52        plusPetiteLongueur = longueurCourante;
53      }
54    }
55    return plusPetitePile;
56  }
57
58
59  PDNNN_PileDeNNN TC_trouverPlusCourtChemin(LAB_Labyrinthe laby){
60    unsigned int premiereCase = CD_obtenirCase(LAB_caseDEntree(
          laby));
61    EDPDNNN_EnsembleDePileDeNNN cheminsPossibles =
          EDPDNNN_ensemble();;
62    PDNNN_PileDeNNN cheminLePlusCourt = PDNNN_pile();
63
64    PDNNN_PileDeNNN cheminCourant = PDNNN_pile();
65    PDNNN_empiler(&cheminCourant,premiereCase);
66    TC_trouverLesChemins(laby,&cheminCourant,&cheminsPossibles);
67    cheminLePlusCourt = TC_comparaisonChemins(cheminsPossibles);
68    return cheminLePlusCourt;
69  }
```

```
1   #include <errno.h>
2   #include <stdio.h>
3   #include <assert.h>
4   #include <stdbool.h>
5   #include "../../include/Types/Ordre.h"
6   #include "../../include/Types/Direction.h"
7   #include "../../include/Types/caseEtDirection.h"
8   #include "../../include/Types/labyrinthe.h"
9   #include "../../include/Types/PileDeNNN.h"
10  #include "../../include/Types/ListeDOrdre.h"
11  #include "../../include/Types/ListeDeCaseEtDirection.h"
12  #include "../../include/Types/EnsembleDeDirection.h"
13
14
15  void CEO_choixOrdre(LAB_Labyrinthe laby, Direction
        directionCasePrecedente, Direction directionCaseActuelle,
        unsigned int caseActuelle, LDO_ListeDOrdre *ordres) {
16    Ordre o;
17    EDD_EnsembleDeDirection directionsPossibles;
18    Direction d;
19    bool estIntersection = false;
20
21    if (directionCasePrecedente == directionCaseActuelle) {
22      directionsPossibles = LAB_directionsPossibles(laby,
            caseActuelle);
23      while (!(EDD_cardinalite(directionsPossibles) == 0)) {
24        d = EDD_obtenirPremierElement(directionsPossibles);
25        EDD_retirer(&directionsPossibles, d);
```

```
26        switch (directionCaseActuelle) {
27        case H :
28          if (!((d == H) || (d == B))) {
29            estIntersection = true;
30          }
31          break;
32        case B :
33          if (!((d == H) || (d == B))) {
34            estIntersection = true;
35          }
36          break;
37        case G :
38          if (!((d == G) || (d == D))) {
39            estIntersection = true;
40          }
41          break;
42        case D :
43          if (!((d == G) || (d == D))) {
44            estIntersection = true;
45          }
46          break;
47        }
48      }
49      if (estIntersection) {
50        LDO_inserer(ordres, AV, LDO_longueur(*ordres) + 1);
51      }
52    }
53    else {
54      switch (directionCasePrecedente) {
55      case H :
56        if  (directionCaseActuelle == G) {
57          o = TG;
58        }
59        else {
60          o = TD;
61        }
62        break;
63      case B :
64        if  (directionCaseActuelle == D) {
65          o = TG;
66        }
67        else {
68          o = TD;
69        }
70        break;
71      case G :
72        if  (directionCaseActuelle == B) {
73          o = TG;
74        }
75        else {
76          o = TD;
77        }
78        break;
79      case D :
```

```
80        if  (directionCaseActuelle == H) {
81          o = TG;
82        }
83        else {
84          o = TD;
85        }
86        break;
87      }
88      LDO_inserer(ordres, o, LDO_longueur(*ordres) + 1);
89      LDO_inserer(ordres, AV, LDO_longueur(*ordres) + 1);
90    }
91  }
92
93
94  Direction CEO_positionPorteEnDirection(Direction d) {
95    Direction res;
96
97    switch (d) {
98    case H :
99      res = B;
100     break;
101   case B :
102     res = H;
103     break;
104   case G :
105     res = D;
106     break;
107   case D:
108     res = G;
109     break;
110   }
111   return res;
112 }
113
114
115 LDCD_ListeDeCaseEtDirection
        CEO_conversionCasesEnCasesEtDirections(LAB_Labyrinthe laby,
         PDNNN_PileDeNNN p) {
116   LDCD_ListeDeCaseEtDirection casesDirections = LDCD_liste();
117   unsigned int caseActuelle, casePrecedente;
118   Direction porteEntree, porteSortie;
119   PDNNN_PileDeNNN copiePile = PDNNN_copier(p);
120
121   caseActuelle = PDNNN_obtenirElement(copiePile);
122   PDNNN_depiler(&copiePile);
123   LAB_DirectionEntreeEtSortie(laby, &porteEntree, &porteSortie)
          ;
124   LDCD_inserer(&casesDirections, CD_caseEtDirection(
          caseActuelle, porteSortie), 1);
125   while (PDNNN_longueur(copiePile) >= 1) {
126     casePrecedente = PDNNN_obtenirElement(copiePile);
127     if (casePrecedente < caseActuelle) {
128       if (casePrecedente + 1 == caseActuelle) {
129         LDCD_inserer(&casesDirections, CD_caseEtDirection(
```

```
            casePrecedente, D), 1);
      }
      else {
        LDCD_inserer(&casesDirections, CD_caseEtDirection(
            casePrecedente, B), 1);
      }
    }
    else {
      if (casePrecedente - 1 == caseActuelle) {
        LDCD_inserer(&casesDirections, CD_caseEtDirection(
            casePrecedente, G), 1);
      }
      else {
        LDCD_inserer(&casesDirections, CD_caseEtDirection(
            casePrecedente, H), 1);
      }
    }
    PDNNN_depiler(&copiePile);

    caseActuelle = casePrecedente;
  }
  return casesDirections;
}


LDO_ListeDOrdre CEO_conversionCasesEtDirectionsEnOrdres(
    LDCD_ListeDeCaseEtDirection casesDirections, LAB_Labyrinthe
     laby) {
  LDO_ListeDOrdre ordres = LDO_liste();
  Direction directionCaseActuelle, directionCasePrecedente,
      porteEntree, porteSortie;
  unsigned int caseActuelle, iLecture;


  iLecture = 1;
  LDO_inserer(&ordres, AV, LDO_longueur(ordres)+1);
  LAB_DirectionEntreeEtSortie(laby, &porteEntree, &porteSortie)
      ;
  directionCasePrecedente = CEO_positionPorteEnDirection(
      porteEntree);
  while (iLecture <= LDCD_longueur(casesDirections)) {
    directionCaseActuelle = CD_obtenirDirection(
        LDCD_obtenirElement(casesDirections, iLecture));
    caseActuelle = CD_obtenirCase(LDCD_obtenirElement(
        casesDirections, iLecture));
    iLecture = iLecture + 1;
    CEO_choixOrdre(laby, directionCasePrecedente,
        directionCaseActuelle, caseActuelle, &ordres);
    directionCasePrecedente = directionCaseActuelle;
  }
  // LDO_inserer(&ordres, ., LDO_longueur(ordres)+1);
  return ordres;
}
```

```c
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <stdbool.h>
#include <string.h>
#include "../../include/Labyrinthe/gestionFichier.h"
#include "../../include/Types/ListeDeCDC.h"
#include "../../include/Types/Direction.h"

void GF_gestionDuFichier(char* nomFichier, unsigned int *
    largeur, unsigned int* caseEntree, unsigned int *caseSortie
    , Direction *directionPorteEntree, Direction *
    directionPorteSortie, LDCDC_ListeDeCDC *listeLiaisons){
  char* modeDOuverture = "r";
  FILE* fichier = fopen(nomFichier,modeDOuverture);

  unsigned int val = 0;
  unsigned int i = 0;
  int nombreDeCaractereMaxAlire = 30;
  char* point = ".";
  if (fichier != NULL){

    char chaine[30] ="";
    bool EstunPoint = false;

    fgets(chaine,nombreDeCaractereMaxAlire,fichier);
    *largeur = (unsigned int)atoi(chaine);

    fgets(chaine,nombreDeCaractereMaxAlire,fichier);
    while (!(chaine[i] == 'D' || chaine[i] == 'G' || chaine[i]
        == 'H' || chaine[i] == 'B' )){
      val = val*10 + (chaine[i]-'0');
      i++;
    }
    *caseEntree = val;
    switch (chaine[i]){
        case 'D':
          *directionPorteEntree = D;
          break;
        case 'G':
          *directionPorteEntree = G;
          break;
        case 'H':
          *directionPorteEntree = H;
          break;
        case 'B':
          *directionPorteEntree = B;
          break;
      }

    val = 0;
    i=0;

```

```
50      fgets(chaine,nombreDeCaractereMaxAlire,fichier);
51      while (!(chaine[i] == 'D' || chaine[i] == 'G' || chaine[i]
            == 'H' || chaine[i] == 'B' )){
52        val = val*10 + (chaine[i]-'0');
53        i++;
54      }
55      *caseSortie = val;
56      switch (chaine[i]){
57          case 'D':
58            *directionPorteSortie = D;
59            break;
60          case 'G':
61            *directionPorteSortie = G;
62            break;
63          case 'H':
64            *directionPorteSortie = H;
65            break;
66          case 'B':
67            *directionPorteSortie = B;
68            break;
69      }
70
71
72      while (fgets(chaine,nombreDeCaractereMaxAlire,fichier) !=
            NULL && !(EstunPoint)){
73        if (chaine[0] == point[0]){
74          EstunPoint = true;
75        }
76        else {
77          LDCDC_inserer(listeLiaisons,chaine,LDCDC_longueur(*
                listeLiaisons)+1);
78        }
79      }
80      fclose(fichier);
81
82    }
83    else{
84      errno = 1;
85      printf("fichier␣vide␣ou␣inexistant");
86    }
87  }
```

```
1   #include <errno.h>
2   #include <assert.h>
3   #include <string.h>
4   #include <stdlib.h>
5   #include <stdio.h>
6
7   #include "../../include/Labyrinthe/initialisationLabyrinthe.h"
8   #include "../../include/Types/Direction.h"
9   #include "../../include/Types/ListeDeCDC.h"
10  #include "../../include/Types/labyrinthe.h"
11
12
```

```
13 LAB_Labyrinthe IL_creerLabyrinthe(unsigned int
       largeurLabyrinthe, unsigned int entree, unsigned int sortie
       , Direction directionPorteEntree, Direction
       directionPorteSortie, LDCDC_ListeDeCDC liaisonsCases){
14   LAB_Labyrinthe laby = LAB_labyrinthe(largeurLabyrinthe,
         entree, sortie, directionPorteEntree,
         directionPorteSortie);
15   IL_creerPassage(&laby, liaisonsCases);
16   return laby;
17 }
18
19 void IL_creerPassage(LAB_Labyrinthe* laby, LDCDC_ListeDeCDC
       liaisonsCases){
20   unsigned int i;
21
22   for (i=1; i<=LDCDC_longueur(liaisonsCases); i++) {
23     char* element = LDCDC_obtenirElement(liaisonsCases, i);
24     unsigned int case1;
25     unsigned int case2;
26     IL_separationCases(element,&case1,&case2);
27     LAB_casserMur(laby,case1,case2);
28   }
29 }
30
31 void IL_separationCases(char* casesASeparer,unsigned int *case1
       ,unsigned int *case2){
32
33   unsigned int milieu;
34   int i=0;
35   while (!(casesASeparer[i] == '-')) {
36     milieu = i+1;
37     i++;
38   }
39   char case1CDC[5]="";
40   char case2CDC[5]="";
41
42   for(int j=0;j<milieu;j++){
43     case1CDC[j] = casesASeparer[j];
44   }
45   for(int p=milieu+1;p<strlen(casesASeparer)-1;p++){
46     case2CDC[p-milieu-1] = casesASeparer[p];
47   }
48
49   *case1 = atoi(case1CDC);
50   *case2 = atoi(case2CDC);
51 }
```

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5 #include <fcntl.h>
6 #include <errno.h>
7 #include <wiringPiI2C.h>
```

```c
#include <linux/input.h>
#include <wiringPi.h>
#include <softPwm.h>
#include <sys/time.h>
#include <lcd.h>

#include <pthread.h>

#include "../include/elec/ecran.h"
#include "../include/elec/codeMoteur.h"
#include "../include/elec/capteurObstacle.h"
#include "../include/elec/capteurLigne.h"
#include "../include/elec/buzz.h"
#include "../include/elec/setup.h"
#include "../include/elec/parcoursLabyrinthe.h"


#include "../include/Types/ListeDOrdre.h"


LDO_ListeDOrdre recupererOrdres(){
  char lOrdre[3];
    int i = 1;
    LDO_ListeDOrdre laListe;

    laListe = LDO_liste();
    printf("La liste des ordres pour arriver à la sortie est : \n");
    do {
        scanf("%2s", lOrdre);

        if (strcmp(lOrdre, "AV") == 0) {
            LDO_inserer(&laListe, AV, i);
            printf("AV\n");
            i++;
        } else if (strcmp(lOrdre, "TG") == 0) {
            LDO_inserer(&laListe, TG, i);
            printf("TG\n");
            i++;
        } else if (strcmp(lOrdre, "TD") == 0) {
            LDO_inserer(&laListe, TD, i);
            printf("TD\n");
            i++;
        }
    }
    while (lOrdre[0] != '.');

    return laListe;
}


int main(int argc ,char **argv){

  LDO_ListeDOrdre listeOrdres1= recupererOrdres();
```

```
61      if (wiringPiSetup()==-1){
62                  printf("Problème d'initialization\n") ;
63      }//init wiringpi
64
65      SET_setupBranchement();
66
67      int lcdHandle = LCD_initLcd();//init écran lcd
68
69      PL_parcoursLabyrinthe(51, lcdHandle, listeOrdres1);
70
71      CM_setupMoteur(80,80,1,1,1);
72
73      return 0;
74  }
```

7.3.3 Les .c de la partie électronique

Ci-dessous sont présentés les .c associés au code du robot.

```
1   /**
2    * @file buzz.c
3    * @brief Spécifie les fonctions du buzzer
4    */
5   #include <lcd.h>
6   #include <wiringPiI2C.h>
7   #include <wiringPi.h>
8   #include <stdio.h>
9   #include <stdlib.h>
10  #include "../../include/elec/buzz.h"
11  #include "../../include/elec/setup.h"
12
13  /**
14   * @brief BUZZ_buzz permet d'activer le buzzer
15   */
16  void BUZZ_buzz (){
17          digitalWrite(BUZZ,1);//acitve le buzzeur
18          }
19
20  /**
21   * @brief BUZZ_debuzz permet de désactiver le buzzer
22   */
23  void BUZZ_debuzz (){
24          digitalWrite(BUZZ,0);//desactive le buzzeur
25          }
```

```
1   /**
2    * @file capteurLigne.c
3    * @brief Spécifie les fonctions analysant les lignes sous le
         robot.
4    */
5   #include <lcd.h>
6   #include <wiringPiI2C.h>
```

```c
#include <wiringPi.h>
#include <stdio.h>
#include <stdlib.h>
#include "../../include/elec/capteurLigne.h"
#include "../../include/elec/setup.h"

/**
 * @brief CL_detectionIntersection permet de savoir si on a détecté une intersection.
 * @return renvoie 1 si une intersection est detecté 0 sinon.
 *
 */
int CL_detectionIntersection() {
        if ((digitalRead(capteurAv) == 1) && ((digitalRead(capteurG) == 1) || (digitalRead(capteurD) == 1))) {
                return 1;
        }
        else {
                return 0;
        }
}

/**
 * @brief CL_sortieLabyrinthe permet de savoir si on a détecté que le robot est sortie du labyrinthe.
 * @return renvoie 1 si le robot est bien sortie 0 sinon.
 *
 */
int CL_sortieLabyrinthe() {
        if ((digitalRead(capteurAr) == 0) && (digitalRead(capteurAv) == 0) && (digitalRead(capteurG) == 0) && (digitalRead(capteurD) == 0)) {
                return 1;
        }
        else {
                return 0;
        }
}

/**
 * @brief CL_seRedresser permet de savoir si on a détecté une situation ou le robot doit se redresser, afin de bien suivre la ligne sous celui-ci.
 * @return renvoie 1 si le robot doit se redresser 0 sinon.
 *
 */
int CL_seRedresser() {//a ou exclusif b
        if ((digitalRead(capteurAv)==0) && (((digitalRead(capteurG)==1) && !(digitalRead(capteurD)==1)) || (!(digitalRead(capteurG)==1) && (digitalRead(capteurD)==1)))) {
                return 1;
        }
        else {
```

```
                return 0;
        }
}
```

```
/**
 * @file capteurObstacle.c
 * @brief Spécifie les fonctions relative à la détection d'un mur devant le robot.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <linux/input.h>
#include <wiringPi.h>
#include <softPwm.h>
#include <sys/time.h>
#include "../../include/elec/capteurObstacle.h"
#include "../../include/elec/setup.h"

/**
 * @brief CO_distanceMesure permet de mesurer la distance avec un potenciel obstacle face au robot.
 * @return renvoie la distance entre le potenciel obstacle devant le robot et celui-ci.
 *
 */

float CO_distanceMesure()
{
        struct timeval tv1;
        struct timeval tv2;
        long start, stop;
        float dis;
        digitalWrite(Trig, LOW);
        delayMicroseconds(2);

        digitalWrite(Trig, HIGH);  //produce a pulse
        delayMicroseconds(10);
        digitalWrite(Trig, LOW);
        while(!(digitalRead(Echo) == 1));
        gettimeofday(&tv1, NULL);           //current time
        while(!(digitalRead(Echo) == 0));
        gettimeofday(&tv2, NULL);           //current time
        start = tv1.tv_sec * 1000000 + tv1.tv_usec;
        stop  = tv2.tv_sec * 1000000 + tv2.tv_usec;

        dis = (float)(stop - start) / 1000000 * 34000 / 2;  //count the distance
        //printf("distance = %f",dis);

```

```
47          return dis;  //Calcule la difference entre le moment d'
                émission de l'impulsion
48          // du son avec le moment d'interception afin de
                calculer la distance entre le capteur et l'objet
49  }
```

```
1   /**
2    * @file ecran.c
3    * @brief Spécifie les fonctions relatives à l'écran lcd
4    */
5   #include <lcd.h>
6   #include <wiringPiI2C.h>
7   #include <wiringPi.h>
8   #include <stdio.h>
9   #include <stdlib.h>
10  #include "../../include/elec/ecran.h"
11
12  //constante LCD
13
14  #define         AF_BASE     64
15  #define         AF_RS                   (AF_BASE + 0)
16  #define         AF_RW                   (AF_BASE + 1)
17  #define         AF_E                    (AF_BASE + 2)
18  #define         AF_LED                  (AF_BASE + 3)
19
20  #define         AF_DB4                  (AF_BASE + 4)
21  #define         AF_DB5                  (AF_BASE + 5)
22  #define         AF_DB6                  (AF_BASE + 6)
23  #define         AF_DB7                  (AF_BASE + 7)
24
25
26  /**
27   * @brief LCD_initLcd initialise l'écran lcd
28   *
29   * @return renvoie la variable relative a l'écran lcd
30   *
31   */
32  int LCD_initLcd(){//Initialisation de l'ecran LCD
33      int lcdHandle;
34
35          int i;
36
37      pcf8574Setup(AF_BASE,0x27); //pcf8574 I2C address
38
39      lcdHandle = lcdInit(2, 16, 4, AF_RS, AF_E, AF_DB4,AF_DB5,
            AF_DB6,AF_DB7, 0,0,0,0) ;
40
41      if (lcdHandle < 0)
42      {
43          fprintf (stderr, "lcdInit␣failed\n") ;
44          exit (EXIT_FAILURE) ;
45      }
46
47      for(i=0;i<8;i++) {
```

```
48          pinMode(AF_BASE+i,OUTPUT);
49          digitalWrite(AF_LED, 1);
50          digitalWrite(AF_RW, 0);
51      }
52      return lcdHandle;
53  }
54
55  /**
56   * @brief LCD_afficherAvancer affiche avancer sur l'écran lcd
57   *
58   * @param lcdHandle variable relative a l'écran
59   *
60   */
61  void LCD_afficherAvancer( int lcdHandle) {
62      lcdClear(lcdHandle);
63      lcdPosition(lcdHandle, 0, 1);
64      lcdPuts(lcdHandle, "En␣avant␣!");
65  }
66
67  /**
68   * @brief LCD_afficherVirageDroite affiche Virage à droite sur
        l'écran lcd
69   *
70   * @param lcdHandle variable relative a l'écran
71   *
72   */
73  void LCD_afficherVirageDroite( int lcdHandle) {
74      lcdClear(lcdHandle);
75      lcdPosition(lcdHandle, 0, 1);
76      lcdPuts(lcdHandle, "Virage␣a␣droite");
77  }
78
79  /**
80   * @brief LCD_afficherVirageGauche affiche virage à gauche sur
        l'écran lcd
81   *
82   * @param lcdHandle variable relative a l'écran
83   *
84   */
85  void LCD_afficherVirageGauche( int lcdHandle) {
86      lcdClear(lcdHandle);
87      lcdPosition(lcdHandle, 0, 1);
88      lcdPuts(lcdHandle, "Virage␣a␣gauche");
89  }
90
91  /**
92   * @brief LCD_afficherIntersection affiche Intersection détecté
        e sur l'écran lcd
93   *
94   * @param lcdHandle variable relative a l'écran
95   *
96   */
97  void LCD_afficherIntersection(int lcdHandle) {
98      lcdClear(lcdHandle);
```

```
99      lcdPosition(lcdHandle, 0, 1);
100     lcdPuts(lcdHandle, "Intersection detectee");
101 }
102
103 /**
104  * @brief LCD_afficherArretUrgence affiche Arret d'urgence sur
        l'écran lcd
105  *
106  * @param lcdHandle variable relative a l'écran
107  *
108  */
109 void LCD_afficherArretUrgence(int lcdHandle) {
110     lcdClear(lcdHandle);
111     lcdPosition(lcdHandle, 0, 1);
112     lcdPuts(lcdHandle, "Arret d'urgence");
113 }
114
115 /**
116  * @brief LCD_afficherFin affiche Sortie trouvee! sur l'écran
        lcd
117  *
118  * @param lcdHandle variable relative a l'écran
119  *
120  */
121 void LCD_afficherFin(int lcdHandle){
122     lcdClear(lcdHandle);
123     lcdPosition(lcdHandle, 0, 1);
124     lcdPuts(lcdHandle, "Sortie trouvee !");
125 }
```

```
1  /**
2   * @file codeMoteur.c
3   * @brief Spécifie les différentes fonctions permettant de
        faire bouger le robot.
4   */
5  #include <lcd.h>
6  #include <wiringPiI2C.h>
7  #include <wiringPi.h>
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include "../../include/elec/setup.h"
11 #include "../../include/elec/codeMoteur.h"
12 #include "../../include/elec/capteurLigne.h"
13
14
15
16 /**
17  * @brief CM_rotation permet au robot d'effectuer une rotation
        sur lui-même
18  *
19  * @param sensRotation le sens de la rotation de type Sens
20  * @param vitesse la vitesse du robot
21  * @param obstacle indique si il y a un obstacle sur le chemin
        via un thread associé
```

```
22  *
23  */
24  void CM_rotation(Sens sensRotation, int vitesse, int obstacle)
        {//plus utiliser
25          unsigned int vitesseMoteurG = vitesse-10;
26          unsigned int vitesseMoteurD = vitesse-10;
27          int avanceMoteurG;
28          int avanceMoteurD;
29          if (sensRotation == GAUCHE) {
30                  avanceMoteurG = 0;
31                  avanceMoteurD = 1;
32          }
33          else {
34                  avanceMoteurG = 1;
35                  avanceMoteurD = 0;
36          }
37          while (!(digitalRead(capteurAv)==1) && (obstacle == 0))
                {
38                  CM_setupMoteur(vitesseMoteurG, vitesseMoteurD,
                        avanceMoteurG, avanceMoteurD, 0);
39          }
40          CM_setupMoteur(vitesseMoteurG, vitesseMoteurD,
                avanceMoteurG, avanceMoteurD, 1);
41  }
42
43  /**
44   * @brief CM_virage permet au robot d'effectuer un virage avec
         une roue tournant plus vite que l'autre
45   *
46   * @param sensVirage le sens du virage de type Sens
47   * @param vitesse la vitesse du robot
48   * @param obstacle indique si il y a un obstacle sur le chemin
         via un thread associé
49   *
50   */
51  void CM_virage(Sens sensVirage, int vitesse, int obstacle) {
52          unsigned int vitesseMoteurG;
53          unsigned int vitesseMoteurD;
54          int avanceMoteurG;
55          int avanceMoteurD;
56          if (sensVirage == GAUCHE) {
57                  vitesseMoteurG = vitesse-28;
58                  vitesseMoteurD = vitesse-3;
59                  avanceMoteurG = 1;
60                  avanceMoteurD = 1;
61          }
62          else {
63                  vitesseMoteurG = vitesse-3; //roue exterieur
64                  vitesseMoteurD = vitesse-28;//roue interieur
65                  avanceMoteurG = 1;
66                  avanceMoteurD = 1;
67          }
68          while (!(digitalRead(capteurD)==0 && digitalRead(
                capteurG)==0 && (digitalRead(capteurAv)==1)) && (
```

```
            obstacle == 0) ) {
69              CM_setupMoteur(vitesseMoteurG, vitesseMoteurD,
                    avanceMoteurG, avanceMoteurD, 0);
70      }
71  }
72
73
74  /**
75   * @brief CM_tourner permet au robot de tourner en fesant d'
         abbord un virage puis une rotation
76   *
77   * @param sensTournant le sens du tournant de type Sens
78   * @param vitesse la vitesse du robot
79   * @param obstacle indique si il y a un obstacle sur le chemin
         via un thread associé
80   *
81   */
82  void CM_tourner(Sens sensTournant, int vitesse, int obstacle) {
83          CM_virage(sensTournant, vitesse, obstacle);
84  }
85
86  /**
87   * @brief CM_avancer permet au robot d'avancer
88   *
89   * @param vitesse la vitesse du robot
90   * @param obstacle indique si il y a un obstacle sur le chemin
         via un thread associé
91   *
92   */
93  void CM_avancer(int vitesse, int obstacle) {
94    unsigned int vitesseMoteurG = vitesse;
95    unsigned int vitesseMoteurD = vitesse;
96    int avanceMoteurG = 1;
97    int avanceMoteurD = 1;
98    int stop = 0;
99    Sens sensRedressage;
100   if (CL_seRedresser() == 1) {
101     if (digitalRead(capteurG) == 1) {
102       sensRedressage = GAUCHE;
103           printf("Redressage gauche\n");
104     }
105     else {
106       sensRedressage = DROITE;
107           printf("redressage droite\n");
108     }
109     CM_rotation(sensRedressage, vitesse, obstacle);
110   }
111   else{
112         CM_setupMoteur(vitesseMoteurG, vitesseMoteurD,
              avanceMoteurG, avanceMoteurD, stop);
113   }
114 }
115
116 /**
```

```
117  * @brief CM_setupMoteur permet de donner une instruction aux
         moteurs
118  *
119  * @param vitesseMoteurG la vitesse souhaiter pour le moteur
         gauche
120  * @param vitesseMoteurD la vitesse souhaiter pour le moteur
         droit
121  * @param avanceMoteurG indique 1 si le moteur gauche doit être
          activé, 0 sinon
122  * @param avanceMoteurD indique 1 si le moteur droit doit être
         activé, 0 sinon
123  * @param stop indique 1 si on doit stopper les moteurs, 0
         sinon
124  *
125  */
126 void CM_setupMoteur(unsigned int vitesseMoteurG, unsigned int
        vitesseMoteurD, int avanceMoteurG, int avanceMoteurD, int
        stop) {
127          digitalWrite(moteurGb1, avanceMoteurG && !stop);
128      digitalWrite(moteurGb2, !avanceMoteurG && !stop);
129      digitalWrite(moteurDb1, avanceMoteurD && !stop);
130      digitalWrite(moteurDb2, !avanceMoteurD && !stop);
131      pwmWrite(moteurEnG,(1024*vitesseMoteurG)/100);
132      pwmWrite(moteurEnD,(1024*vitesseMoteurD)/100);
133 }
134
135 void CM_emote(unsigned int vitesse){
136          int k=0;
137          while(k<5){
138          CM_setupMoteur(vitesse,vitesse,1,0,0);
139          delay(100);
140          CM_setupMoteur(vitesse,vitesse,0,1,0);
141          delay(100);
142          k=k+1;
143          }
144          CM_setupMoteur(0,0,1,1,1);
145 }
```

```
1  #include <wiringPi.h>
2
3  #include <pthread.h>
4  #include <stdio.h>
5
6  #include "../../include/elec/parcoursLabyrinthe.h"
7  #include "../../include/elec/capteurLigne.h"
8  #include "../../include/elec/capteurObstacle.h"
9  #include "../../include/elec/buzz.h"
10 #include "../../include/elec/codeMoteur.h"
11 #include "../../include/elec/ecran.h"
12 #include "../../include/elec/setup.h"
13
14 #include "../../include/Types/Ordre.h"
15 #include "../../include/Types/ListeDOrdre.h"
16
```

```
17 int obstacle = 0;//obstacle devant?
18 float distance = 0;//distance mesurer par les capteurs ultrason
19 int situation = 0;// situation actuelle (on avance, on tourne a
       droite ou a gauche, fin du parcours)
20 int lcdHandle = 0;//lcd variable
21 int vitesse = 0;//vitesse du robot
22
23 pthread_mutex_t mutex;
24
25 void* arretDUrgence(void* arg) {
26     while (1) {
27         pthread_mutex_lock(&mutex);  // Protéger l'accès à `
              distance` et `obstacle`
28         distance = CO_distanceMesure();
29         if (distance > 15) {
30             BUZZ_debuzz();
31             obstacle = 0; // Pas d'obstacle
32         } else {
33             CM_setupMoteur(0, 0, 0, 0, 1);
34             BUZZ_buzz();
35             obstacle = 1; // Obstacle détecté
36         }
37         pthread_mutex_unlock(&mutex);  // Libérer le mutex
38         usleep(50000); // Pause de 50 ms pour libérer le CPU
39     }
40     return NULL;
41 }
42
43 void* gestionLCD(void* arg) {
44     int lcdHandle = *(int*)arg;
45
46     while (1) {
47         pthread_mutex_lock(&mutex); // Protéger l'accès aux
              variables partagées
48         int localSituation = situation;
49         int localObstacle = obstacle;
50         float localDistance = distance;
51         pthread_mutex_unlock(&mutex);
52
53         if (localObstacle == 1) {
54             LCD_afficherArretUrgence(lcdHandle);
55         } else {
56             switch (localSituation) {
57                 case 4:
58                     LCD_afficherFin(lcdHandle);
59                     break;
60                 case 3:
61                     LCD_afficherVirageGauche(lcdHandle);
62                     break;
63                 case 2:
64                     LCD_afficherVirageDroite(lcdHandle);
65                     break;
66                 case 1:
67                     LCD_afficherAvancer(lcdHandle);
```

```
68                      break;
69                  case 0:
70                      LCD_afficherIntersection(lcdHandle);
71                      break;
72              }
73          }
74          usleep(500000); // 500 ms
75      }
76      return NULL;
77  }
78
79  /**
80   * @brief PL_parcoursLabyrinthe va parcourir le labyrinthe
         selon une liste d'ordres en utilisant obstacle et
         situation.
81   *
82   * @param vitesseInput vitesse souhaité lors du parcours
83   * @param lcdHandleInput variable relative à l'écran lcd afin
         de l'actualisé au cours du parcours
84   * @param laListeDOrdres une liste d'ordres de type Ordre qui
         nous permettrons de parcourir le labyrinthe
85   *
86   */
87  void PL_parcoursLabyrinthe (int vitesseInput, int
        lcdHandleInput, LDO_ListeDOrdre laListeDOrdres){
88
89    lcdHandle = lcdHandleInput;
90    vitesse = vitesseInput;
91
92    pthread_t threadUrgence;
93    pthread_t threadLCD;
94
95    pthread_mutex_init(&mutex, NULL); // Initialisation du mutex
96    if (pthread_create(&threadUrgence, NULL, arretDUrgence, NULL)
          != 0) {
97          printf("Erreur␣lors␣de␣la␣création␣du␣thread␣
                arretUrgence␣\n");
98          return -1;
99      }
100   printf("Thread␣arretUrgence␣créé␣avec␣succès\n");
101
102   if (pthread_create(&threadLCD, NULL, gestionLCD, &lcdHandle)
          != 0) {
103         printf("Erreur␣lors␣de␣la␣création␣du␣thread␣LCD\n");
104         return -1;
105     }
106   printf("Thread␣LCD␣créé␣avec␣succès\n");
107   Ordre unOrdre;
108   unsigned int i = 1;
109   CM_setupMoteur(0,0,1,1,1);
110   while(i < LDO_longueur(laListeDOrdres)){
111     unOrdre = LDO_obtenirElement(laListeDOrdres,i);
112     printf("ordre␣:␣%d\n␣", i);
113     int avpr = 0;
```

```
114     switch (unOrdre){
115       case AV://on a deja avancer jusqu'a trouver un tournant
116         situation = 1;
117         if(avpr==1){
118           CM_setupMoteur(vitesse,vitesse,1,1,0);
119           delay(500);
120         }
121         printf("Avancer\n");
122         while((digitalRead(capteurD)==1 || digitalRead(capteurG
                )==1)){//on avance pour ne plus detecter l'
                intersection
123           printf("sort inter\n");
124           CM_setupMoteur(vitesse, vitesse, 1, 1, 0);
125         }
126         printf("sortie boucle\n");
127         while(CL_detectionIntersection() != 1){
128           if(obstacle == 0){
129             CM_avancer(vitesse, obstacle);
130           }
131         }
132         avpr = 1;
133         printf("Intersection\n");
134         situation = 0;//intersection
135         break;
136       case TD:
137         CM_setupMoteur(vitesse-2, vitesse-27,1,1,0); //test
138         delay(400); //test
139         printf("Tourner Droite\n");
140         while((digitalRead(capteurAv)==1) || digitalRead(
                capteurD)==1){//on avance pour ne plus detecter l'
                intersection
141           CM_setupMoteur(vitesse-3, vitesse-30, 1, 1, 0);
142         }
143
144         situation = 2;//droite
145         while(digitalRead(capteurAv) != 1){
146           if(obstacle == 0){
147             CM_tourner(DROITE, vitesse, obstacle);
148           }
149         }
150         printf("Intersection\n");
151         situation = 0;//intersection
152         avpr=0;
153         break;
154       case TG:
155         CM_setupMoteur(vitesse-27, vitesse-2,1,1,0); //test
156         delay(400); //test
157         printf("Tourner Gauche\n");
158         while((digitalRead(capteurAv)==1) || digitalRead(
                capteurG)==1){//on avance pour ne plus detecter l'
                intersection
159           CM_setupMoteur(vitesse-30, vitesse-3, 1, 1, 0);
160         }
161         situation = 3;//gauche
```

```
162         while(digitalRead(capteurAv) != 1){
163           if(obstacle == 0){
164             CM_tourner(GAUCHE, vitesse, obstacle);
165           }
166         }
167         situation = 0;//intersection
168         printf("intersection\n");
169         avpr = 0;
170         break;
171       default:
172         break;
173       }
174     i=i+1;
175     CM_setupMoteur(0,0,1,1,1);
176     //delay(5000);
177   }
178   int j = 0;
179   while(j<50000){
180     CM_avancer(vitesse,vitesse,1,1,0);
181     j=j+1;
182   }
183   //CM_setupMoteur(vitesse,vitesse,1,1,0);
184   delay(500);
185   situation = 4;//fin du parcours
186   //CM_emote(vitesse);
187   CM_setupMoteur(0,0,1,1,1);
188
189 }
```

```
1  /**
2   * @file setup.c
3   * @brief Setup tout les composants utiliser
4   */
5  #include <wiringPi.h>
6  #include "../../include/elec/setup.h"
7  /**
8   * @brief SET_setupBranchement permet de setup tout les port de
        la raspberry qu'on va utiliser pour : les moteurs, les
       capteurs de lignes, le buzzer, le capteur d'obstacle.
9   */
10 void SET_setupBranchement() {
11         wiringPiSetupGpio();
12     wiringPiSetup();
13         pinMode(moteurEnG,PWM_OUTPUT);//moteurs
14         pinMode(moteurEnD,PWM_OUTPUT);
15         pinMode(moteurGb1,OUTPUT);
16         pinMode(moteurGb2,OUTPUT);
17         pinMode(moteurDb1,OUTPUT);
18         pinMode(moteurDb2,OUTPUT);
19         pinMode(capteurAr, INPUT);//captligne
20         pinMode(capteurAv, INPUT);
21         pinMode(capteurD, INPUT);
22         pinMode(capteurG, INPUT);
23         pinMode(BUZZ, OUTPUT);//buzzer
```

```
24          pinMode(Echo, INPUT);//captObstacle
25          pinMode(Trig, OUTPUT);
26  }
```

7.4 Tests unitaires

Une fois que toutes les fonctions et procédures sont créées, il faut les tester indépendamment afin de s'assurer de leur fiabilité. C'est une étape très importante, car elle permet de rendre les tests d'intégration bien plus fiables. En effet, si un problème survient lors des tests d'intégration, cela signifie que c'est une erreur d'algorithmie et non d'implémentation. Nous avons donc réalisés pas moins de 124 tests unitaires sur les types.

7.5 Tests d'intégrations

Une fois que nous avions la garantie que toutes les fonctions/ procédures fonctionnaient correctement, nous avons commencé l'étape des tests d'intégration. Celle-ci consiste à mettre en lien chaque fonction et de vérifier que celles-ci nous retournent bien les résultats attendus.

7.5.1 Partie algorithmique

Concernant la partie algorithmique, le but était de vérifier que les fonctions de la sortie du labyrinthe sont corrects. Pour cela, on donne le fichier du labyrinthe à notre programme puis on vérifie que chaque partie du programme donne le résultat attendu. Cela permet de voir et comprendre les erreurs du programme à chaque étape pour ainsi les corriger, notamment en utilisant l'outil de débuggage permettant de s'arrêter à divers moments de l'exécution.

Le but était de pouvoir identifier et de résoudre les erreurs plus rapidement. Ces tests ont été très pertinents car ils ont permis de relever certaines incohérences et identifier des aspects qui ont été mal anticipés durant le projet. Pour illustrer, la copie des piles, des listes, ou des ensembles qui n'avaient pas été réalisés correctement. C'est pour cela qu'il a fallu ajouter des fonctions spécifiques afin de réaliser des deep copies. De plus, nous avons réalisé que nos algorithmes n'étaient pas toujours des plus évidents. Par exemple, l'utilisation d'une pile stockant le chemin courant ainsi qu'un ensemble de pile stockant tous les chemins possibles. Pour trouver le chemin le plus court, il a donc fallu implémenter une fonction qui compare ces chemins. Cependant, après réflexion, il aurait été préférable d'utiliser une liste de cases. Cela nous aurait offert de nombreux avantages :

— L'implémentation du type EnsembleDePileDeNNN n'était plus nécessaire.
— L'implémentation du type PileDeNNN n'était plus nécessaire.
— La manipulation de la Liste est bien plus simple que la manipulation de la Pile.
— L'implémentation de la fonction de comparaison des chemins pouvait être évitée.

Pour conclure, l'algorithme fonctionne parfaitement mais nous avons perçu quelques axes d'améliorations si c'était à refaire.

7.5.2 Partie électronique

Nous avons dû tester chaque composant de manière indépendante, afin de vérifier le bon fonctionnement du composant et de ses fonctions associées. Le test s'est déroulé en 2 parties :
— Le test électrique : le composant était branché à une source d'alimentation, puis à l'aide d'un voltmètre, nous avons effectué des prises de tensions pour vérifier le bon fonctionnement.
— Le test du code C : en gardant le composant branché, on exécute un programme simple pour vérifier si la réponse du composant est bien celle attendue.

Dès lors, nous avons exécuté notre code pour vérifier le bon comportement du composant. Finalement, nous avons regroupé ensemble toutes les fonctions pour créer un algorithme global dans l'objectif de faire sortir le robot du labyrinthe.

Encore une fois, ces tests ont été très pertinents et nous ont permis de prendre conscience de nos erreurs. Dans un premier temps, nous avons réalisé que la stratégie utlisée pour guider le robot au sein du labyrinthe était théoriquement bonne, mais physiquement trop complexe à mettre en place. En effet, la longueur des cases ne nous permettait pas de tourner correctement. Nous avons donc dû trouver une autre approche pour résoudre le labyrinthe. La difficulté résidait alors dans le fait que nous ne pouvions pas changer la position des capteurs à cause de la contrainte de temps. Cependant, la façon dont nous avons implémenté nos fonctions nous permettait de les réutiliser assez facilement dans un autre contexte.

Le positionnement du capteur arrière s'est donc révélé plutôt inutile, ce qui nous a forçé à implementer le code à l'aide de 3 capteurs seulement, une tâche plutôt complexe quand la majorité des groupes utilisent 4 capteurs voire 5. Lorsque nous détections un virage, nous avancions un peu de manière à ne plus le détecter, puis nous tournions jusqu'à ce que nous récupérions la ligne avec le capteur du milieu. Il est intéressant de noter que cette

manière de prendre les virages respecte bien la consigne de suivre les lignes tout en veillant à ne franchir aucun mur, c'est à dire que le robot restait parfaitement dans les cases.

Finalement, bien que nous ayons réussi à trouver une méthode satisfaisante, nous avons rencontré quelques problèmes, notamment au niveau des intersections. Bien que nous sommes parvenus, à faire sortir le robot du labyrinthe proposé dans le sujet, en effectuant la résolution du labyrinthe à l'envers (la case d'entrée devient la case de sortie et vice-versa) nous nous sommes heurtés au problème suivant : le robot considère les successions d'ordres avancer comme un seul ordre avancer. En effet, bien qu'il reçoive les ordres d'avancer les uns à la suite des autres et bien qu'il soit censé poursuivre en ligne droite, le robot tourne. Le problème réside alors dans une entrée de boucle qui n'est pas respectée. Plus précisemment, lorsque le robot détecte une intersection, nous voulons qu'il continue jusqu'à ce qu'il ne la détecte plus, puis qu'il effectue l'ordre suivant. Cependant, celui ci n'attend pas de sortir de l'intersection et enchaine les ordres avancer très rapidement.

Ainsi, lors de la soutenance, notre robot a parfaitement démarré en effectuant les quatre virages de suite, et a finalement rencontré le même problème. Nous sommes tout de même fiers du travail accompli.

Chapitre 8

Conclusion

8.1 Conclusion globale sur le projet

De manière plus générale, ce projet nous a permis d'assimiler et de renforcer de nombreuses compétences. En effet, celui-ci utilise un large pannel d'outils et de méthodes de travail indispensable pour un ingénieur. Bien que cette première expérience n'aie pas toujours été parfaitement réalisée, elle aura été très enrichissante. Elle nous aura permis de nous projeter plus concrètement sur le travail que nous aurons à effectuer, et les difficultés que nous rencontrerons, dans les années à venir.

8.2 Ressenti personnel de chaque membre du projet

— **Delaplace Yohann**
Le projet m'a permis de lier la théorie et la pratique en la mettant en œuvre sur le robot. Cela m'a aussi permis de m'habituer à utiliser Git afin de réaliser un travail en groupe. J'ai eu l'occasion de programmer en C et en Shell, ce qui m'a permis de comprendre plus en profondeur les notions abordées en cours. Concernant le groupe de travail, chacun avait sa tâche assignée par le chef de projet, avec la date limite à ne pas dépasser pour que le projet puisse bien avancer, et chacun d'entre nous a respecté ce fonctionnement, permettant au projet de se développer plus rapidement. De plus, je tiens à souligner la bonne entente dans notre groupe de travail, ce qui nous a permis de rester motivés du début à la fin et de tout donner pour réussir et avoir un projet qui fonctionne. Je pense que si ce projet a pu être mené à bien, c'est grâce au grand investissement personnel de chacun.

— **Lenoble Louis**

Bien que le robot ne soit pas encore totalement opérationnel, je suis fier du chemin parcouru par le groupe et de la manière dont chacun a su trouver sa place. Cette complémentarité a créé une dynamique positive et productive, où les idées de chacun enrichissaient le projet. J'ai particulièrement apprécié l'engagement collectif et l'entraide qui se sont développés au fil des séances.

Ensuite, ce projet a profondément changé ma manière de percevoir le travail en groupe. J'ai appris à mieux écouter, à anticiper les besoins des autres et à m'adapter aux imprévus. De plus, j'ai pris conscience que l'organisation rigoureuse et la communication ouverte ont été d'une très grande importance pour garantir un avancement régulier et efficace. C'est un apprentissage que je considère précieux, et je me sens désormais bien mieux préparé pour travailler de manière collaborative dans d'autres contextes.

Enfin, en tant que chef de projet, cette expérience a été révélatrice. Je me suis rendu compte que ce rôle demande bien plus qu'une simple coordination : il s'agit d'être à l'écoute, de motiver, et de savoir tirer le meilleur de chaque membre du groupe. Organiser les séances, planifier les objectifs et faire des bilans réguliers m'ont demandé beaucoup de temps, mais cela m'a permis de développer une meilleure capacité d'analyse et de décision. Cette immersion dans la gestion de projet m'a permis de grandir, à la fois en tant que leader et en tant qu'individu capable de travailler de façon autonome.

— **Planchot Maël**

— **Sanson Dylan**

— **Sourdrille Nathan**
Ce projet était une première grande expérience pour moi de part la quantité de travail à fournir, l'étendu de la durée du projet, l'utilisation de nouveaux outils et le travail en groupe de cinq. D'abord, la quantité de travail à fournir était très importante, et cela bien que nous ayons réparti, il me semble, plutôt bien le travail. Ce projet s'est étendu sur deux mois mais quelques jours de plus aurait été grandement apprécié, afin de résoudre les problèmes rencontrés lors de la phase de test, que nous n'avons pu effectuer que durant la dernière semaine. D'autre part, je n'avais jamais utilisé, hormis leur introduction durant les séances de td, les outils gitlab, latex ... C'était aussi une première expérience de vrai program-

mation en C. Cependant c'était très intéressant d'exploiter ce que nous avons pu voir dans les différentes matières, et de pousser plus loin la SE par exemple, en naviguant dans le terminal, créant des scripts bash ... Globalement, le projet est très intéressant et très satisfaisant quand on arrive finalement à faire sortir le robot du labyrinthe, cependant il demande énormément de travail personnel, de l'organisation et une entente harmonieuse entre les membres du groupe est indispensable pour mener le projet à son terme.

Chapitre 9

Annexes

9.1 Références bibliographiques

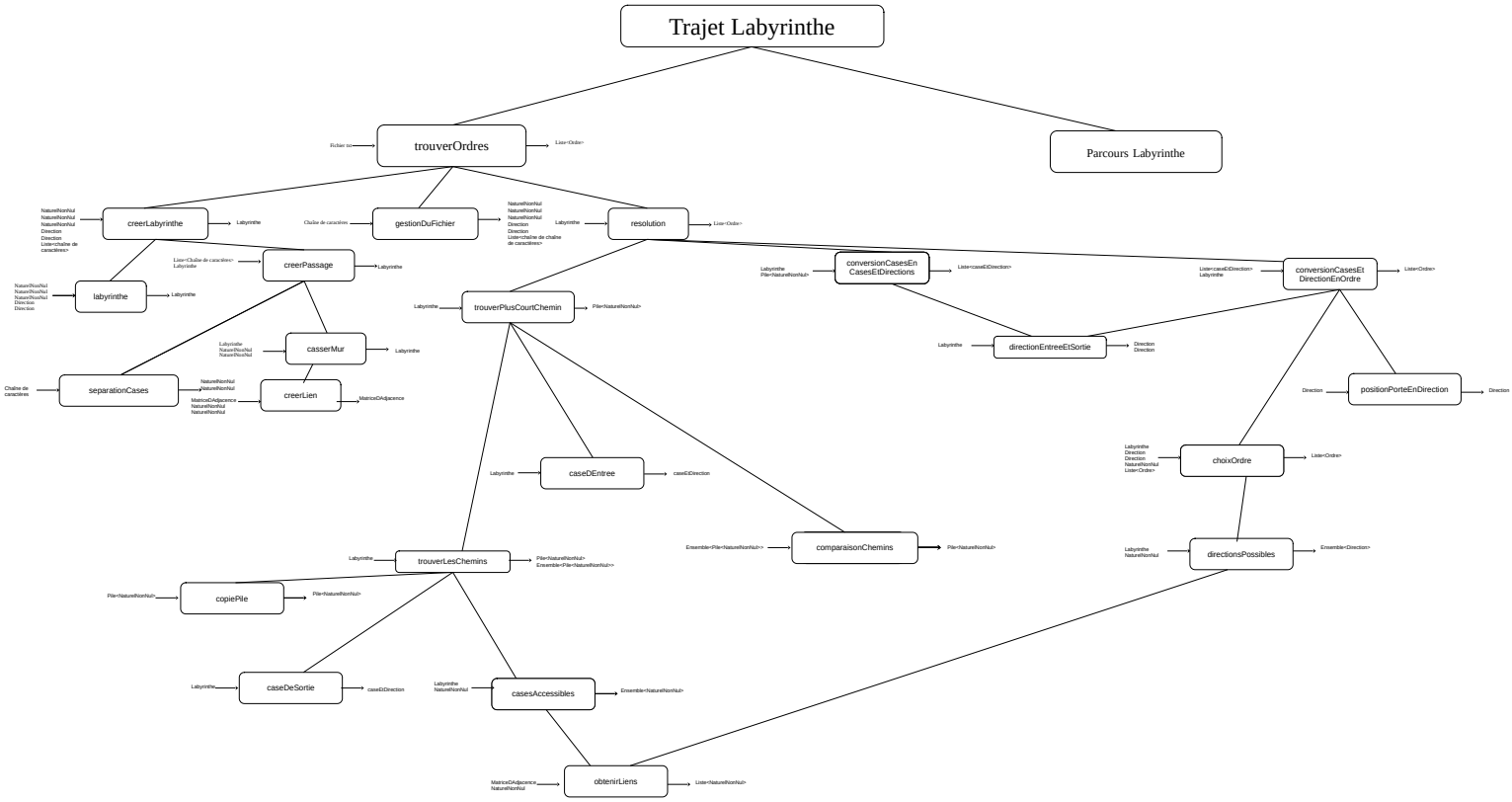
— • https://cdn.bodanius.com/media/1/a9c1593_fiche-technique-du-tcrt5000.pdf pour les capteurs de positions
— • https://www.adeept.com/ultrasonic-sr04_p0047.html pour le capteur a ultrason (récupération de la tension et du courant d'entrée necessaire pour déterminer la puissance)
— • https://cdn.bodanius.com/media/1/8c2164078_l293d.pdf pour les bouclier thermiques (carte moteur)
— • https://www.openhacks.com/uploadsproductos/eone-1602a1.pdf pour l'écran LCD et le controleur de l'écran (référence dans le tableau section : 3.0ELECTRICAL CHARACTERISTICS)
— • https://moodle.insa-rouen.fr/pluginfile.php/184607/mod_resource/content/5/Lab_Sheet__1.pdf pour la LED suivis de
— • https://www.adeept.com/4pcsled_p0104.html pour la la LED (documentation et informations manquantes pour le sujet, on utilise donc le cas général pour une led rouge)
— • https://www.adeept.com/passivebuzzer_p0284.html pour le buzzer, la documentation est manquante, on prendra la cas "général" de la consomation énergétique d'un buzzer.
— • https://www.adeept.com/2xn20-with-holder_p0335.html pour les moteurs, consommation maximale en ampère prise en compte.
— • https://moodle.insa-rouen.fr/pluginfile.php/31526/mod_label/intro/documentation.pdf pour la documentation Latex

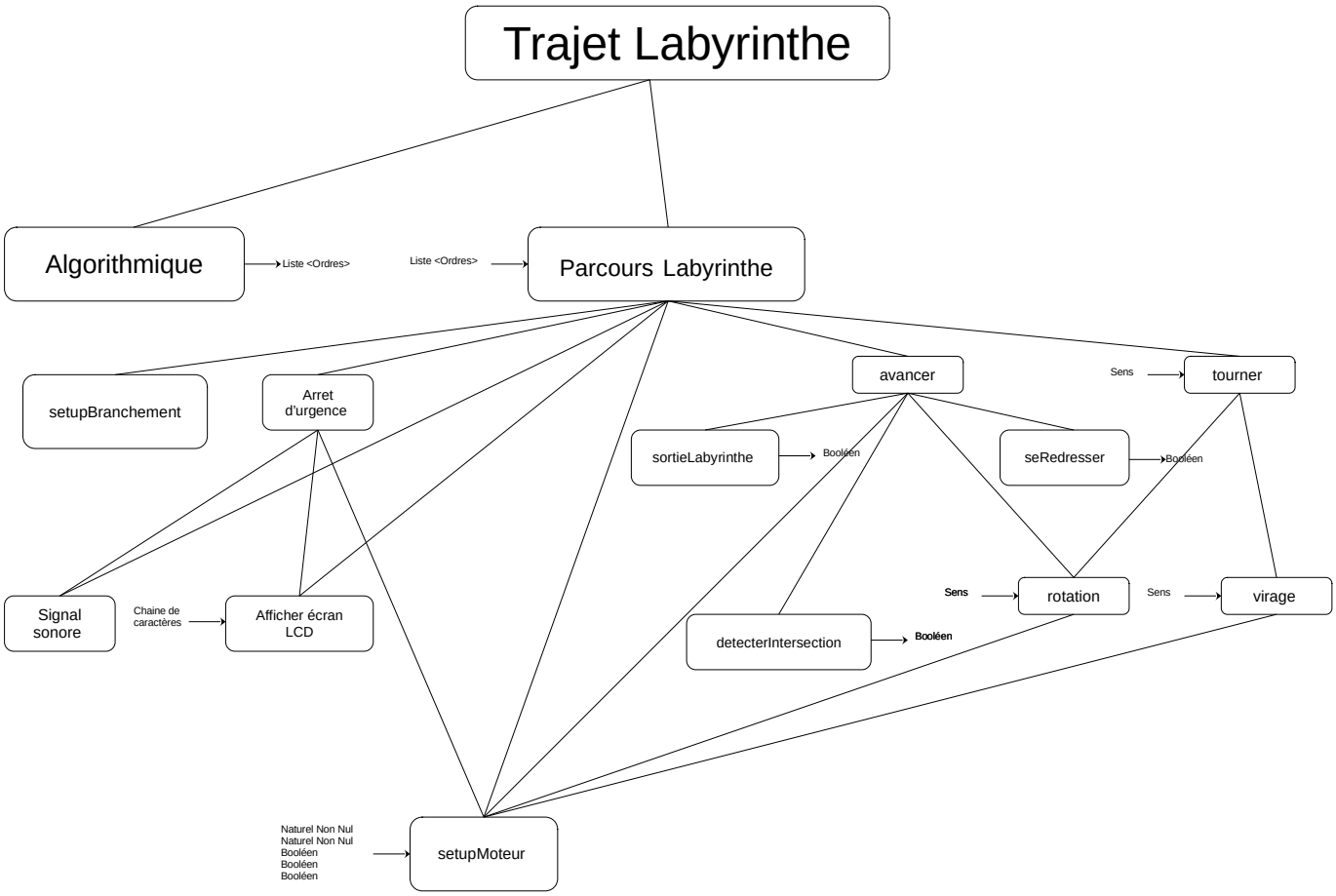
9.2 Glossaire

Nom	abréviation	Définition
TAD	Type Abstrait de Données	Structure de données définie par des opérations (est indépendant du language de programmation).
GPIO	General Purpose Input/Output	Une broche de la carte Raspberry qui peut être configurée comme une entée pour reçevoir un signal ou comme une sortie pour reçevoir un signal.
SDA	Serial Data Line	Ligne de commmunication utilisé dans les bus I2C utilisée pour envoyer et reçevoir des données.
SCL	Serial Clock Line	Ligne de communication utilisé dans les bus I2C utilisée pour synchoniser la transmition de données (définit l'orloge).
I2C	Inter-Integrated Circuit	Protocole de communication entre composants.
GND	Ground	Masse du circuit électronique.
VCC	Voltage at the Common Collector	Source de tension positive d'alimentation d'un circuit électronique.
Trig	Trigger	Ligne utilisée pour le signal de déclanchement d'un capteur.
Tx	Transmit	Ligne utilisée pour la transmition de données.
Rx	Receive	Ligne utilisée pour la réception de données.
Echo	écho	Signal reçu du capteur à ultrason après réverbération du son sur une surface.
LED	Light Emitting Diode	Composant électronique qui emmet de la lumière lorsque un certain seuil de tension est dépassé (consomme moins d'électricité qu'une lampe classique).
PWM	Pulse Width Modulation	Le signal envoyé peut changer la durée des impulsions.
SD	Secure Digital	carte mémoire pour stocker des données.
SSH	Secure Shell	Protocole de communication crypté pour se connexter à distance sur un autre appareil.
RAM	Random Acces Memory	Mémoire en accès direct, stocke temporairement des données.
EN	Enable	Ligne utilisée pour activer ou désactiver un circuit électronique.
LCD	Liquid Crystal Display	Permet de crontrôler la quantité de lumière qui passe à travers de l'écran à l'aide d'un champ électrique.

9.3 Analyses descendente version PDF

Ci dessous sont présentés en format page entière les analyses descendantes mises en place lors du cycle en V.


Trajet Labyrinthe
trouverOrdres
Fichier txt
Liste<Ordre>
Parcours Labyrinthe
creerLabyrinthe
NatureNonNul
NatureNonNul
NatureNonNul
Direction
Direction
Liste<chaîne de caractères>
Labyrinthe
gestionDuFichier
Chaîne de caractères
resolution
NatureNonNul
NatureNonNul
NatureNonNul
Direction
Direction
Liste<chaîne de chaîne de caractères>
Labyrinthe
Liste<Ordre>
creerPassage
Liste<Chaîne de caractères>
Labyrinthe
Labyrinthe
labyrinthe
NatureNonNul
NatureNonNul
NatureNonNul
Direction
Direction
Labyrinthe
conversionCasesEn CasesEtDirections
Labyrinthe
Pile<NatureNonNul>
Liste<caseEtDirection>
conversionCasesEt DirectionEnOrdre
Liste<caseEtDirection>
Labyrinthe
Liste<Ordre>
trouverPlusCourtChemin
Labyrinthe
Pile<NatureNonNul>
directionEntreeEtSortie
Labyrinthe
Direction
Direction
casserMur
Labyrinthe
NatureNonNul
NatureNonNul
Labyrinthe
separationCases
Chaîne de caractères
NatureNonNul
NatureNonNul
creerLien
MatriceDAdjacence
NatureNonNul
NatureNonNul
MatriceDAdjacence
positionPorteEnDirection
Direction
Direction
caseDEntree
Labyrinthe
caseEtDirection
choixOrdre
Labyrinthe
Direction
Direction
NatureNonNul
Liste<Ordre>
Liste<Ordre>
comparaisonChemins
Ensemble<Pile<NatureNonNul>>
Pile<NatureNonNul>
trouverLesChemins
Labyrinthe
Pile<NatureNonNul>
Ensemble<Pile<NatureNonNul>>
directionsPossibles
Labyrinthe
NatureNonNul
Ensemble<Direction>
copiePile
Pile<NatureNonNul>
Pile<NatureNonNul>
caseDeSortie
Labyrinthe
caseEtDirection
casesAccessibles
Labyrinthe
NatureNonNul
Ensemble<NatureNonNul>
obtenirLiens
MatriceDAdjacence<
NatureNonNul
Liste<NatureNonNul>


Trajet Labyrinthe
Algorithmique
Liste <Ordres>
Liste <Ordres>
Parcours Labyrinthe
setupBranchement
Arret d'urgence
avancer
Sens
tourner
sortieLabyrinthe
Booléen
seRedresser
Booléen
Signal sonore
Chaine de caractères
Afficher écran LCD
Sens
rotation
Sens
virage
detecterIntersection
Booléen
Naturel Non Nul
Naturel Non Nul
Booléen
Booléen
Booléen
setupMoteur

9.4 Fichiers .c

9.4.1 Les Types

Les listes

```
#include <stdbool.h>
#include <stddef.h>
#include <assert.h>
#include <errno.h>
#include <stdlib.h>
#include "../../include/Types/ListeChaineeDeNNN.h"
#include "../../include/Types/ListeDeNNN.h"

LDNNN_ListeDeNNN LDNNN_liste(){
        LDNNN_ListeDeNNN liste;
        liste.lesElements = LCDNNN_listeVide();
        liste.nbElements = 0;
        return liste;
}


bool LDNNN_estVide(LDNNN_ListeDeNNN liste){
        return liste.nbElements == 0;
}


void LDNNN_inserer(LDNNN_ListeDeNNN *liste,
    unsigned int element, unsigned int position){

  if (position == 1) {
    LCDNNN_ListeChaineeDeNNN nouveauNoeud = (
        LCDNNN_ListeChaineeDeNNN)malloc(sizeof(
        LCDNNN_Noeud));
    nouveauNoeud->element = element;
    nouveauNoeud->listeSuivante = liste->
        lesElements;
    liste->lesElements = nouveauNoeud;
    liste->nbElements++;
    return;
  }
  else{
    LCDNNN_ListeChaineeDeNNN courant = liste->
        lesElements;
    for (unsigned int i = 1; i < position - 1; i
        ++) {
      courant = courant->listeSuivante;
    }

```

```
38      LCDNNN_ListeChaineeDeNNN nouveauNoeud = (
            LCDNNN_ListeChaineeDeNNN)malloc(sizeof(
            LCDNNN_Noeud));
39      nouveauNoeud->element = element;
40      nouveauNoeud->listeSuivante = courant->
            listeSuivante;
41
42      courant->listeSuivante = nouveauNoeud;
43      liste->nbElements++;
44    }
45  }
46
47
48  void LDNNN_supprimer(LDNNN_ListeDeNNN *liste,
        unsigned int position){
49    if (position == 1) {
50      LCDNNN_ListeChaineeDeNNN aSupprimer = liste->
            lesElements;
51      liste->lesElements = aSupprimer->listeSuivante
            ;
52      free(aSupprimer);
53    } else {
54      LCDNNN_ListeChaineeDeNNN courant = liste->
            lesElements;
55      for (unsigned int i = 1; i < position - 1; i
            ++) {
56        courant = courant->listeSuivante;
57      }
58
59      LCDNNN_ListeChaineeDeNNN aSupprimer = courant
            ->listeSuivante;
60      courant->listeSuivante = aSupprimer->
            listeSuivante;
61      free(aSupprimer);
62    }
63    liste->nbElements--;
64  }
65
66
67
68  unsigned int LDNNN_obtenirElement(LDNNN_ListeDeNNN
        liste, unsigned int position){
69    LCDNNN_ListeChaineeDeNNN courant = liste.
          lesElements;
70    for (unsigned int i = 1; i < position; i++) {
71      courant = courant->listeSuivante;
72    }
73    return courant->element;
74  }
```

```
75
76 unsigned int LDNNN_longueur(LDNNN_ListeDeNNN liste
       ){
77         return liste.nbElements;
78 }
```

```
1  #include <stdbool.h>
2  #include <stddef.h>
3  #include <assert.h>
4  #include <errno.h>
5  #include <stdlib.h>
6  #include "../../include/Types/ListeChaineeDOrdre.h
       "
7  #include "../../include/Types/ListeDOrdre.h"
8
9  LDO_ListeDOrdre LDO_liste() {
10   LDO_ListeDOrdre liste;
11   liste.lesElements = LCDO_listeVide();
12   liste.nbElements = 0;
13   return liste;
14 }
15
16 bool LDO_estVide(LDO_ListeDOrdre liste) {
17   return liste.nbElements == 0;
18 }
19
20
21 void LDO_inserer(LDO_ListeDOrdre *liste, Ordre
       element, unsigned int position) {
22
23   if (position == 1) {
24     LCDO_ajouter(&(liste->lesElements), element);
25     liste->nbElements++;
26   }
27   else {
28     LCDO_ListeChaineeDOrdre courant = liste->
         lesElements;
29     for (unsigned int i = 1; i < position - 1; i
         ++) {
30       courant = courant->listeSuivante;
31     }
32
33     LCDO_Noeud *nouveauNoeud = (LCDO_Noeud*)malloc
         (sizeof(LCDO_Noeud));
34     nouveauNoeud->element = element;
35     nouveauNoeud->listeSuivante = courant->
         listeSuivante;
36     courant->listeSuivante = nouveauNoeud;
37
```

```
38      liste->nbElements++;
39    }
40  }
41
42
43  void LDO_supprimer(LDO_ListeDOrdre *liste,
        unsigned int position) {
44
45
46    if (position == 1) {
47      LCDO_supprimerTete(&(liste->lesElements));
48    }
49    else {
50      LCDO_ListeChaineeDOrdre courant = liste->
            lesElements;
51      for (unsigned int i = 1; i < position - 1; i
            ++) {
52        courant = courant->listeSuivante;
53      }
54
55      LCDO_Noeud *aSupprimer = courant->
            listeSuivante;
56      courant->listeSuivante = aSupprimer->
            listeSuivante;
57      free(aSupprimer);
58
59      liste->nbElements--;
60    }
61  }
62
63
64  Ordre LDO_obtenirElement(LDO_ListeDOrdre liste,
        unsigned int position) {
65    LCDO_ListeChaineeDOrdre courant = liste.
          lesElements;
66    for (unsigned int i = 1; i < position; i++) {
67      courant = courant->listeSuivante;
68    }
69    return courant->element;
70  }
71
72  unsigned int LDO_longueur(LDO_ListeDOrdre liste) {
73    return liste.nbElements;
74  }
```

```
1  #include <stdbool.h>
2  #include <stddef.h>
3  #include <assert.h>
4  #include <errno.h>
```

```c
#include <stdlib.h>
#include <string.h>
#include "../../include/Types/ListeDeCDC.h"
#include "../../include/Types/ListeChaineeDeCDC.h"


LDCDC_ListeDeCDC LDCDC_liste(){
  LDCDC_ListeDeCDC liste;
  liste.lesElements = LCDCDC_listeVide();
  liste.nbElements = 0;
  return liste;
}


bool LDCDC_estVide(LDCDC_ListeDeCDC liste){
  return liste.nbElements == 0;
}


void LDCDC_inserer(LDCDC_ListeDeCDC *liste, char *
    element, unsigned int position){

  if (position == 1) {
    LCDCDC_ajouter(&liste->lesElements, element);
  }
  else {
    LCDCDC_ListeChaineeDeCDC courant = liste->
        lesElements;
    for (unsigned int i = 1; i < position - 1; i
        ++) {
      courant = courant->listeSuivante;
    }
    LCDCDC_ListeChaineeDeCDC nouveauNoeud = (
        LCDCDC_ListeChaineeDeCDC)malloc(sizeof(
        LCDCDC_Noeud));
    strcpy(nouveauNoeud->element, element);
    nouveauNoeud->listeSuivante = courant->
        listeSuivante;
    courant->listeSuivante = nouveauNoeud;
  }
  liste->nbElements++;
}


void LDCDC_supprimer(LDCDC_ListeDeCDC *liste,
    unsigned int position){
  if (position == 1) {
    LCDCDC_supprimerTete(&liste->lesElements);
  }
```

```
47    else {
48      LCDCDC_ListeChaineeDeCDC courant = liste->
            lesElements;
49      for (unsigned int i = 1; i < position - 1; i
            ++) {
50        courant = courant->listeSuivante;
51      }
52      LCDCDC_ListeChaineeDeCDC aSupprimer = courant
            ->listeSuivante;
53      courant->listeSuivante = aSupprimer->
            listeSuivante;
54      free(aSupprimer);
55    }
56    liste->nbElements--;
57  }
58
59
60  char* LDCDC_obtenirElement(LDCDC_ListeDeCDC liste,
        unsigned int position){
61    LCDCDC_ListeChaineeDeCDC courant = liste.
          lesElements;
62    for (unsigned int i = 1; i < position; i++) {
63      courant = courant->listeSuivante;
64    }
65    return courant->element;
66  }
67
68
69  unsigned int LDCDC_longueur(LDCDC_ListeDeCDC liste
        ){
70    return liste.nbElements;
71  }
```

```
1   #include <stdbool.h>
2   #include <stddef.h>
3   #include <assert.h>
4   #include <errno.h>
5   #include <stdlib.h>
6   #include "../../include/Types/caseEtDirection.h"
7   #include "../../include/Types/
        ListeChaineeDeCaseEtDirection.h"
8   #include "../../include/Types/
        ListeDeCaseEtDirection.h"
9
10
11  LDCD_ListeDeCaseEtDirection LDCD_liste() {
12      LDCD_ListeDeCaseEtDirection liste;
13      liste.lesElements = LCDCD_listeVide();
14      liste.nbElements = 0;
```

```
15      return liste;
16  }
17
18
19  bool LDCD_estVide(LDCD_ListeDeCaseEtDirection
        liste) {
20      return liste.nbElements == 0;
21  }
22
23
24  void LDCD_inserer(LDCD_ListeDeCaseEtDirection *
        liste, CD_CaseEtDirection element, unsigned int
         position) {
25      if (position == 1) {
26          LCDCD_ListeChaineeDeCaseEtDirection
                nouveauNoeud = (
                LCDCD_ListeChaineeDeCaseEtDirection)
                malloc(sizeof(LCDCD_Noeud));
27          nouveauNoeud->element = element;
28          nouveauNoeud->listeSuivante = liste->
                lesElements;
29          liste->lesElements = nouveauNoeud;
30          liste->nbElements++;
31          return;
32      }
33      else {
34          LCDCD_ListeChaineeDeCaseEtDirection
                courant = liste->lesElements;
35          for (unsigned int i = 1; i < position - 1;
                 i++) {
36              courant = courant->listeSuivante;
37          }
38
39          LCDCD_ListeChaineeDeCaseEtDirection
                nouveauNoeud = (
                LCDCD_ListeChaineeDeCaseEtDirection)
                malloc(sizeof(LCDCD_Noeud));
40          nouveauNoeud->element = element;
41          nouveauNoeud->listeSuivante = courant->
                listeSuivante;
42
43          courant->listeSuivante = nouveauNoeud;
44          liste->nbElements++;
45      }
46  }
47
48
49  void LDCD_supprimer(LDCD_ListeDeCaseEtDirection *
        liste, unsigned int position) {
```

```
50      if (position == 1) {
51          LCDCD_ListeChaineeDeCaseEtDirection
                aSupprimer = liste->lesElements;
52          liste->lesElements = aSupprimer->
                listeSuivante;
53          free(aSupprimer);
54      }
55      else {
56          LCDCD_ListeChaineeDeCaseEtDirection
                courant = liste->lesElements;
57          for (unsigned int i = 1; i < position - 1;
                 i++) {
58              courant = courant->listeSuivante;
59          }
60
61          LCDCD_ListeChaineeDeCaseEtDirection
                aSupprimer = courant->listeSuivante;
62          courant->listeSuivante = aSupprimer->
                listeSuivante;
63          free(aSupprimer);
64      }
65      liste->nbElements--;
66  }
67
68
69  CD_CaseEtDirection LDCD_obtenirElement(
        LDCD_ListeDeCaseEtDirection liste, unsigned int
         position) {
70      LCDCD_ListeChaineeDeCaseEtDirection courant =
            liste.lesElements;
71      for (unsigned int i = 1; i < position; i++) {
72          courant = courant->listeSuivante;
73      }
74      return courant->element;
75  }
76
77  unsigned int LDCD_longueur(
        LDCD_ListeDeCaseEtDirection liste) {
78      return liste.nbElements;
79  }
```

Les listes chaînées

```
1  #include <stdbool.h>
2  #include <stddef.h>
3  #include <assert.h>
4  #include <errno.h>
5  #include <stdlib.h>
```

```
6  #include "../../include/Types/ListeChaineeDeNNN.h"
7
8
9  LCDNNN_ListeChaineeDeNNN LCDNNN_listeVide(){
10         errno=0;
11         return NULL;
12 }
13
14
15 bool LCDNNN_estVide(LCDNNN_ListeChaineeDeNNN
       uneListe){
16         errno=0;
17         return uneListe == NULL;
18 }
19
20 void LCDNNN_ajouter(LCDNNN_ListeChaineeDeNNN *
       uneListe, unsigned int element){
21         LCDNNN_ListeChaineeDeNNN pNoeud = (
              LCDNNN_ListeChaineeDeNNN)malloc(sizeof(
              LCDNNN_Noeud));
22         if (pNoeud!=NULL){
23         errno = 0;
24         pNoeud -> element = element;
25         pNoeud -> listeSuivante = *uneListe;
26         *uneListe = pNoeud;
27         }
28         else{
29         errno=LCDNNN_ERREUR_MEMOIRE;
30         }
31 }
32
33
34 unsigned int LCDNNN_obtenirElement(
       LCDNNN_ListeChaineeDeNNN uneListe){
35         assert(!LCDNNN_estVide(uneListe));
36         errno=0;
37         return uneListe->element;
38 }
39
40 void LCDNNN_fixerElement(LCDNNN_ListeChaineeDeNNN
       *uneListe, unsigned int element){
41         assert(!LCDNNN_estVide(*uneListe));
42         errno=0;
43         (*uneListe)->element = element;
44 }
45
46 LCDNNN_ListeChaineeDeNNN
       LCDNNN_obtenirListeSuivante(
       LCDNNN_ListeChaineeDeNNN uneListe){
```

```
47          assert(!LCDNNN_estVide(uneListe));
48          errno=0;
49          return uneListe->listeSuivante;
50  }
51
52  void LCDNNN_fixerListeSuivante(
        LCDNNN_ListeChaineeDeNNN *uneListe,
        LCDNNN_ListeChaineeDeNNN nelleSuite){
53          assert(!LCDNNN_estVide(*uneListe));
54          errno=0;
55          (*uneListe)->listeSuivante = nelleSuite;
56  }
57
58  void LCDNNN_supprimerTete(LCDNNN_ListeChaineeDeNNN
        *uneListe){
59          LCDNNN_ListeChaineeDeNNN temp;
60          assert(!LCDNNN_estVide(*uneListe));
61          errno=0;
62          temp=*uneListe;
63          *uneListe = LCDNNN_obtenirListeSuivante(*
                uneListe);
64          free(temp);
65  }
66
67  void LCDNNN_supprimer(LCDNNN_ListeChaineeDeNNN *
        uneListe){
68          errno=0;
69          if (!LCDNNN_estVide(*uneListe)){
70                  LCDNNN_supprimerTete(uneListe);
71                  LCDNNN_supprimer(uneListe);
72          }
73  }
74
75  void LCDNNN_concatener(LCDNNN_ListeChaineeDeNNN *
        liste1, LCDNNN_ListeChaineeDeNNN liste2) {
76    LCDNNN_ListeChaineeDeNNN temp;
77
78    if (LCDNNN_estVide(*liste1)) {
79      *liste1 = liste2;
80    }
81    else {
82      if (!LCDNNN_estVide(liste2)) {
83        temp = LCDNNN_obtenirListeSuivante(*liste1);
84        LCDNNN_concatener(&temp, liste2);
85        if (LCDNNN_estVide(
              LCDNNN_obtenirListeSuivante(*liste1))) {
86          LCDNNN_fixerListeSuivante(liste1, temp);
87        }
88      }
```

```
  }
}



void LCDNNN_inverser(LCDNNN_ListeChaineeDeNNN *
    uneliste){
  LCDNNN_ListeChaineeDeNNN temp;

  if (!LCDNNN_estVide(*uneliste)) {
    temp = LCDNNN_obtenirListeSuivante(*uneliste);
    LCDNNN_inverser(&temp);
    LCDNNN_fixerListeSuivante(uneliste,
        LCDNNN_listeVide());
    LCDNNN_concatener(&temp, *uneliste);
    *uneliste = temp;
  }
}




LCDNNN_ListeChaineeDeNNN LCDNNN_copier(
    LCDNNN_ListeChaineeDeNNN uneliste){
  LCDNNN_ListeChaineeDeNNN copie= LCDNNN_listeVide
      ();
  LCDNNN_ListeChaineeDeNNN temp = uneliste;

  while (!LCDNNN_estVide(temp)) {
    LCDNNN_ajouter(&copie, LCDNNN_obtenirElement(
        temp));
    temp = LCDNNN_obtenirListeSuivante(temp);
  }
  LCDNNN_inverser(&copie);

  return copie;
}
```

```
#include <stdbool.h>
#include <stddef.h>
#include <assert.h>
#include <errno.h>
#include <stdlib.h>
#include "../../include/Types/Ordre.h"
#include "../../include/Types/ListeChaineeDOrdre.h
    "


LCDO_ListeChaineeDOrdre LCDO_listeVide(){
```

```
11          errno=0;
12          return NULL;
13  }
14
15
16  bool LCDO_estVide(LCDO_ListeChaineeDOrdre uneListe
        ){
17          errno=0;
18          return uneListe == NULL;
19  }
20
21  void LCDO_ajouter(LCDO_ListeChaineeDOrdre *
        uneListe, Ordre element){
22          LCDO_ListeChaineeDOrdre pNoeud = (
                LCDO_ListeChaineeDOrdre )malloc(sizeof(
                LCDO_Noeud));
23          if (pNoeud!=NULL){
24          errno = 0;
25          pNoeud -> element = element;
26          pNoeud -> listeSuivante = *uneListe;
27          *uneListe = pNoeud;
28          }
29          else{
30          errno= LCDO_ERREUR_MEMOIRE;
31          }
32  }
33
34
35  unsigned int LCDO_obtenirElement(
        LCDO_ListeChaineeDOrdre uneListe){
36          assert(!LCDO_estVide(uneListe));
37          errno=0;
38          return uneListe->element;
39  }
40
41  void LCDO_fixerElement(LCDO_ListeChaineeDOrdre *
        uneListe, Ordre element){
42          assert(!LCDO_estVide(*uneListe));
43          errno=0;
44          (*uneListe)->element = element;
45  }
46
47  LCDO_ListeChaineeDOrdre LCDO_obtenirListeSuivante(
        LCDO_ListeChaineeDOrdre uneListe){
48          assert(!LCDO_estVide(uneListe));
49          errno=0;
50          return uneListe->listeSuivante;
51  }
52
```

```
53 void LCDO_fixerListeSuivante(
       LCDO_ListeChaineeDOrdre *uneListe,
       LCDO_ListeChaineeDOrdre nelleSuite){
54         assert(!LCDO_estVide(*uneListe));
55         errno=0;
56         (*uneListe)->listeSuivante = nelleSuite;
57 }
58
59
60 void LCDO_supprimerTete(LCDO_ListeChaineeDOrdre *
       uneListe){
61         LCDO_ListeChaineeDOrdre temp;
62         assert(!LCDO_estVide(*uneListe));
63         errno=0;
64         temp=*uneListe;
65         *uneListe = LCDO_obtenirListeSuivante(*
              uneListe);
66         free(temp);
67 }
68
69 void LCDO_supprimer(LCDO_ListeChaineeDOrdre *
       uneListe){
70         errno=0;
71         if (!LCDO_estVide(*uneListe)){
72                 LCDO_supprimerTete(uneListe);
73                 LCDO_supprimer(uneListe);
74         }
75 }
```

```
1  #include <stdbool.h>
2  #include <stddef.h>
3  #include <assert.h>
4  #include <errno.h>
5  #include <stdlib.h>
6  #include <string.h>
7  #include "../../include/Types/ListeChaineeDeCDC.h"
8
9  LCDCDC_ListeChaineeDeCDC LCDCDC_listeVide(){
10         errno=0;
11         return NULL;
12 }
13
14 bool LCDCDC_estVide(LCDCDC_ListeChaineeDeCDC
       uneListe){
15         errno=0;
16         return uneListe == NULL;
17 }
18
19 void LCDCDC_ajouter(LCDCDC_ListeChaineeDeCDC *
```

```
    uneListe, char *element){
20        LCDCDC_ListeChaineeDeCDC pNoeud = (
              LCDCDC_ListeChaineeDeCDC)malloc(sizeof(
              LCDCDC_Noeud));
21        if (pNoeud!=NULL){
22                errno = 0;
23                strcpy(pNoeud->element, element);
24                pNoeud -> listeSuivante = *
                      uneListe;
25                *uneListe = pNoeud;
26        }
27        else{
28                errno=LCDCDC_ERREUR_MEMOIRE;
29        }
30  }
31
32
33  char *LCDCDC_obtenirElement(
      LCDCDC_ListeChaineeDeCDC uneListe){ /* assert :
       estVide(uneListe) */
34        assert(!LCDCDC_estVide(uneListe));
35        errno=0;
36        return uneListe->element;
37  }
38
39  void LCDCDC_fixerElement(LCDCDC_ListeChaineeDeCDC
      *uneListe, char *element){ /* assert : estVide(
      uneListe) */
40        assert(!LCDCDC_estVide(*uneListe));
41        errno=0;
42        strcpy((*uneListe)->element, element);
43  }
44
45
46  LCDCDC_ListeChaineeDeCDC
      LCDCDC_obtenirListeSuivante(
      LCDCDC_ListeChaineeDeCDC uneListe){
47        assert(!LCDCDC_estVide(uneListe));
48        errno=0;
49        return uneListe->listeSuivante;
50  }
51
52
53  void LCDCDC_fixerListeSuivante(
      LCDCDC_ListeChaineeDeCDC *uneListe,
      LCDCDC_ListeChaineeDeCDC nelleSuite){
54        assert(!LCDCDC_estVide(*uneListe));
55        errno=0;
56        (*uneListe)->listeSuivante = nelleSuite;
```

```
57 }
58
59 void LCDCDC_supprimerTete(LCDCDC_ListeChaineeDeCDC
       *uneListe){ /* assert : estVide(uneListe) */
60         LCDCDC_ListeChaineeDeCDC temp;
61         assert(!LCDCDC_estVide(*uneListe));
62         errno=0;
63         temp=*uneListe;
64         *uneListe = LCDCDC_obtenirListeSuivante(*
               uneListe);
65         free(temp);
66 }
67
68 void LCDCDC_supprimer(LCDCDC_ListeChaineeDeCDC *
       uneListe){
69         errno=0;
70         if (!LCDCDC_estVide(*uneListe)){
71                 LCDCDC_supprimerTete(uneListe);
72                 LCDCDC_supprimer(uneListe);
73         }
74 }
```

```
1  #include <stdbool.h>
2  #include <stddef.h>
3  #include <assert.h>
4  #include <errno.h>
5  #include <stdlib.h>
6  #include "../../include/Types/caseEtDirection.h"
7  #include "../../include/Types/
       ListeChaineeDeCaseEtDirection.h"
8
9
10 LCDCD_ListeChaineeDeCaseEtDirection
       LCDCD_listeVide() {
11     errno = 0;
12     return NULL;
13 }
14
15
16 bool LCDCD_estVide(
       LCDCD_ListeChaineeDeCaseEtDirection uneListe) {
17     errno = 0;
18     return uneListe == NULL;
19 }
20
21
22 void LCDCD_ajouter(
       LCDCD_ListeChaineeDeCaseEtDirection *uneListe,
       CD_CaseEtDirection element) {
```

```
23      LCDCD_ListeChaineeDeCaseEtDirection pNoeud = (
            LCDCD_ListeChaineeDeCaseEtDirection)malloc(
            sizeof(LCDCD_Noeud));
24      if (pNoeud != NULL) {
25          errno = 0;
26          pNoeud->element = element;
27          pNoeud->listeSuivante = *uneListe;
28          *uneListe = pNoeud;
29      } else {
30          errno = LCDCD_ERREUR_MEMOIRE;
31      }
32  }
33
34
35  CD_CaseEtDirection LCDCD_obtenirElement(
        LCDCD_ListeChaineeDeCaseEtDirection uneListe) {
36      assert(!LCDCD_estVide(uneListe));
37      errno = 0;
38      return uneListe->element;
39  }
40
41
42  void LCDCD_fixerElement(
        LCDCD_ListeChaineeDeCaseEtDirection *uneListe,
        CD_CaseEtDirection element) {
43      assert(!LCDCD_estVide(*uneListe));
44      errno = 0;
45      (*uneListe)->element = element;
46  }
47
48
49  LCDCD_ListeChaineeDeCaseEtDirection
        LCDCD_obtenirListeSuivante(
        LCDCD_ListeChaineeDeCaseEtDirection uneListe) {
50      assert(!LCDCD_estVide(uneListe));
51      errno = 0;
52      return uneListe->listeSuivante;
53  }
54
55
56  void LCDCD_fixerListeSuivante(
        LCDCD_ListeChaineeDeCaseEtDirection *uneListe,
        LCDCD_ListeChaineeDeCaseEtDirection nelleSuite)
         {
57      assert(!LCDCD_estVide(*uneListe));
58      errno = 0;
59      (*uneListe)->listeSuivante = nelleSuite;
60  }
61
```

```
62
63 void LCDCD_supprimerTete(
       LCDCD_ListeChaineeDeCaseEtDirection *uneListe)
   {
64     LCDCD_ListeChaineeDeCaseEtDirection temp;
65     assert(!LCDCD_estVide(*uneListe));
66     errno = 0;
67     temp = *uneListe;
68     *uneListe = LCDCD_obtenirListeSuivante(*
           uneListe);
69     free(temp);
70 }
71
72
73 void LCDCD_supprimer(
       LCDCD_ListeChaineeDeCaseEtDirection *uneListe)
   {
74     errno = 0;
75     if (!LCDCD_estVide(*uneListe)) {
76         LCDCD_supprimerTete(uneListe);
77         LCDCD_supprimer(uneListe);
78     }
79 }
```

```
1  #include <stdbool.h>
2  #include <stdio.h>
3  #include <stddef.h>
4  #include <assert.h>
5  #include <errno.h>
6  #include <stdlib.h>
7  #include "../../include/Types/
       ListeChaineeDeDirection.h"
8  #include "../../include/Types/Direction.h"
9
10
11 LCDD_ListeChaineeDeDirection LCDD_listeVide(){
12         errno=0;
13         return NULL;
14 }
15
16
17
18 bool LCDD_estVide(LCDD_ListeChaineeDeDirection
       uneListe){
19         errno=0;
20         return uneListe == NULL;
21 }
22
23
```

```
24 void LCDD_ajouter(LCDD_ListeChaineeDeDirection *
       uneListe, Direction element){
25                 LCDD_ListeChaineeDeDirection
                       pNoeud = (
                       LCDD_ListeChaineeDeDirection)
                       malloc(sizeof(LCDD_Noeud));
26         if (pNoeud!=NULL){
27         errno = 0;
28         pNoeud -> element = element;
29         pNoeud -> listeSuivante = *uneListe;
30         *uneListe = pNoeud;
31         }
32         else{
33         errno=LCDD_ERREUR_MEMOIRE;
34         }
35 }
36
37
38 Direction LCDD_obtenirElement(
       LCDD_ListeChaineeDeDirection uneListe){
39   Direction d;
40         assert(!LCDD_estVide(uneListe));
41         errno=0;
42         d = uneListe->element;
43         return d;
44 }
45
46
47 void LCDD_fixerElement(
       LCDD_ListeChaineeDeDirection *uneListe,
       Direction element){
48         assert(!LCDD_estVide(*uneListe));
49         errno=0;
50         (*uneListe)->element = element;
51 }
52
53 LCDD_ListeChaineeDeDirection
       LCDD_obtenirListeSuivante(
       LCDD_ListeChaineeDeDirection uneListe){
54         assert(!LCDD_estVide(uneListe));
55         errno=0;
56         return uneListe->listeSuivante;
57 }
58
59 void LCDD_fixerListeSuivante(
       LCDD_ListeChaineeDeDirection *uneListe,
       LCDD_ListeChaineeDeDirection nelleSuite){
60         assert(!LCDD_estVide(*uneListe));
61         errno=0;
```

```
62          (*uneListe)->listeSuivante = nelleSuite;
63  }
64
65  void LCDD_supprimerTete(
        LCDD_ListeChaineeDeDirection *uneListe){
66          LCDD_ListeChaineeDeDirection temp;
67          assert(!LCDD_estVide(*uneListe));
68          errno=0;
69          temp=*uneListe;
70          *uneListe = LCDD_obtenirListeSuivante(*
                uneListe);
71          free(temp);
72  }
73
74  void LCDD_supprimer(LCDD_ListeChaineeDeDirection *
        uneListe){
75          errno=0;
76          if (!LCDD_estVide(*uneListe)){
77                  LCDD_supprimerTete(uneListe);
78                  LCDD_supprimer(uneListe);
79          }
80  }
```

```
1   #include <stdbool.h>
2   #include <stddef.h>
3   #include <assert.h>
4   #include <errno.h>
5   #include <stdlib.h>
6   #include "../../include/Types/PileDeNNN.h"
7   #include "../../include/Types/
        ListeChaineeDePileDeNNN.h"
8
9
10  LCDPDNNN_ListeChaineeDePileDeNNN
        LCDPDNNN_listeVide(){
11          errno=0;
12          return NULL;
13  }
14
15
16  bool LCDPDNNN_estVide(
        LCDPDNNN_ListeChaineeDePileDeNNN uneListe){
17          errno=0;
18          return uneListe == NULL;
19  }
20
21
22  void LCDPDNNN_ajouter(
        LCDPDNNN_ListeChaineeDePileDeNNN *uneListe,
```

```
    PDNNN_PileDeNNN element){
23        LCDPDNNN_ListeChaineeDePileDeNNN pNoeud =
              (LCDPDNNN_ListeChaineeDePileDeNNN)
              malloc(sizeof(LCDPDNNN_Noeud));
24        if (pNoeud!=NULL){
25        errno = 0;
26        pNoeud -> element = element;
27        pNoeud -> listeSuivante = *uneListe;
28        *uneListe = pNoeud;
29        }
30        else{
31        errno= LCDPDNNN_ERREUR_MEMOIRE;
32        }
33  }
34
35
36  PDNNN_PileDeNNN LCDPDNNN_obtenirElement(
      LCDPDNNN_ListeChaineeDePileDeNNN uneListe){
37        assert(!LCDPDNNN_estVide(uneListe));
38        errno=0;
39        return uneListe->element;
40  }
41
42  void LCDPDNNN_fixerElement(
      LCDPDNNN_ListeChaineeDePileDeNNN *uneListe,
      PDNNN_PileDeNNN element){
43        assert(!LCDPDNNN_estVide(*uneListe));
44        errno=0;
45        (*uneListe)->element = element;
46  }
47
48  LCDPDNNN_ListeChaineeDePileDeNNN
      LCDPDNNN_obtenirListeSuivante(
      LCDPDNNN_ListeChaineeDePileDeNNN uneListe){
49        assert(!LCDPDNNN_estVide(uneListe));
50        errno=0;
51        return uneListe->listeSuivante;
52  }
53
54  void LCDPDNNN_fixerListeSuivante(
      LCDPDNNN_ListeChaineeDePileDeNNN *uneListe,
      LCDPDNNN_ListeChaineeDePileDeNNN nelleSuite){
      /* assert : estVide(uneListe) */
55        assert(!LCDPDNNN_estVide(*uneListe));
56        errno=0;
57        (*uneListe)->listeSuivante = nelleSuite;
58  }
59
60  void LCDPDNNN_supprimerTete(
```

```
    LCDPDNNN_ListeChaineeDePileDeNNN *uneListe){ /*
     assert : estVide(uneListe) */
61          LCDPDNNN_ListeChaineeDePileDeNNN temp;
62          assert(!LCDPDNNN_estVide(*uneListe));
63          errno=0;
64          temp=*uneListe;
65          *uneListe = LCDPDNNN_obtenirListeSuivante
                (*uneListe);
66          free(temp);
67  }
68
69
70  void LCDPDNNN_supprimer(
        LCDPDNNN_ListeChaineeDePileDeNNN *uneListe){
71          errno=0;
72          if (!LCDPDNNN_estVide(*uneListe)){
73                  LCDPDNNN_supprimerTete(uneListe);
74                  LCDPDNNN_supprimer(uneListe);
75          }
76  }
77
78
79  void LCDPDNNN_concatener(
        LCDPDNNN_ListeChaineeDePileDeNNN *liste1,
        LCDPDNNN_ListeChaineeDePileDeNNN liste2) {
80    LCDPDNNN_ListeChaineeDePileDeNNN temp;
81
82    if (LCDPDNNN_estVide(*liste1)) {
83      *liste1 = liste2;
84    }
85    else {
86      if (!LCDPDNNN_estVide(liste2)) {
87        temp = LCDPDNNN_obtenirListeSuivante(*liste1
              );
88        LCDPDNNN_concatener(&temp, liste2);
89        if (LCDPDNNN_estVide(
              LCDPDNNN_obtenirListeSuivante(*liste1)))
              {
90          LCDPDNNN_fixerListeSuivante(liste1, temp);
91        }
92      }
93    }
94  }
95
96
97
98  void LCDPDNNN_inverser(
        LCDPDNNN_ListeChaineeDePileDeNNN *uneliste){
99    LCDPDNNN_ListeChaineeDePileDeNNN temp;
```

```
100
101   if (!LCDPDNNN_estVide(*uneliste)) {
102     temp = LCDPDNNN_obtenirListeSuivante(*uneliste
          );
103     LCDPDNNN_inverser(&temp);
104     LCDPDNNN_fixerListeSuivante(uneliste,
          LCDPDNNN_listeVide());
105     LCDPDNNN_concatener(&temp, *uneliste);
106     *uneliste = temp;
107   }
108 }
109
110
111
112
113 LCDPDNNN_ListeChaineeDePileDeNNN LCDPDNNN_copier(
      LCDPDNNN_ListeChaineeDePileDeNNN uneliste){
114   LCDPDNNN_ListeChaineeDePileDeNNN copie=
        LCDPDNNN_listeVide();
115   LCDPDNNN_ListeChaineeDePileDeNNN temp = uneliste
        ;
116
117   while (!LCDPDNNN_estVide(temp)) {
118     LCDPDNNN_ajouter(&copie,
          LCDPDNNN_obtenirElement(temp));
119     temp = LCDPDNNN_obtenirListeSuivante(temp);
120   }
121   LCDPDNNN_inverser(&copie);
122
123   return copie;
124 }
```

Les ensembles

```
1  #include <stdbool.h>
2  #include <stddef.h>
3  #include <assert.h>
4  #include <errno.h>
5  #include <stdlib.h>
6  #include "../../include/Types/ListeChaineeDeNNN.h"
7  #include "../../include/Types/EnsembleDeNNN.h"
8
9  EDNNN_EnsembleDeNNN EDNNN_ensemble(){
10         EDNNN_EnsembleDeNNN ensemble;
11         ensemble.lesElements = LCDNNN_listeVide();
12         ensemble.nbElements = 0;
13         return ensemble;
14 }
```

```
15
16 void EDNNN_ajouter(EDNNN_EnsembleDeNNN *unEnsemble
       , unsigned int element){
17         if (!EDNNN_estPresent(*unEnsemble, element
               )){
18                 LCDNNN_ajouter(&unEnsemble->
                       lesElements, element);
19                 unEnsemble->nbElements++;
20         }
21 }
22
23 void EDNNN_retirer(EDNNN_EnsembleDeNNN *unEnsemble
       , unsigned int element){
24         LCDNNN_ListeChaineeDeNNN *noeudCourant =
               &(unEnsemble->lesElements);
25         LCDNNN_ListeChaineeDeNNN precedent = NULL;
26         bool elementRetire = false;
27
28         while (!LCDNNN_estVide(*noeudCourant) && !
               elementRetire) {
29                 if (LCDNNN_obtenirElement(*
                       noeudCourant) == element) {
30                         if (precedent == NULL) {
31                                 LCDNNN_supprimerTete
                                       (noeudCourant);
32                         }
33                         else {
34                                 LCDNNN_fixerListeSuivante
                                       (&precedent,
                                       LCDNNN_obtenirListeSuivante
                                       (*noeudCourant)
                                       );
35                                 free(*noeudCourant
                                       );
36                                 *noeudCourant =
                                       NULL;
37                         }
38                         unEnsemble->nbElements--;
39                         elementRetire = true;
40                         }
41                 else {
42                         precedent = *noeudCourant;
43                         noeudCourant = &(*
                               noeudCourant)->
                               listeSuivante;
44                 }
45         }
46 }
47
```

```
48
49  bool EDNNN_estPresent(EDNNN_EnsembleDeNNN
        unEnsemble, unsigned int element){
50          LCDNNN_ListeChaineeDeNNN noeudCourant =
                unEnsemble.lesElements;
51           bool estPresent = false;
52
53          while (!LCDNNN_estVide(noeudCourant) && !
                estPresent) {
54                  if (LCDNNN_obtenirElement(
                        noeudCourant) == element) {
55                          estPresent = true;
56          }
57          noeudCourant = LCDNNN_obtenirListeSuivante
                (noeudCourant);
58          }
59
60      return estPresent;
61  }
62
63  unsigned int EDNNN_cardinalite(EDNNN_EnsembleDeNNN
        unEnsemble){
64
65          return unEnsemble.nbElements;
66  }
67
68
69  EDNNN_EnsembleDeNNN EDNNN_unionEnsemble(
        EDNNN_EnsembleDeNNN e1, EDNNN_EnsembleDeNNN e2)
        {
70          EDNNN_EnsembleDeNNN unionEnsemble =
                EDNNN_ensemble();
71
72          LCDNNN_ListeChaineeDeNNN noeudCourant = e1
                .lesElements;
73          while (!LCDNNN_estVide(noeudCourant)) {
74                  EDNNN_ajouter(&unionEnsemble,
                        LCDNNN_obtenirElement(
                        noeudCourant));
75                  noeudCourant =
                        LCDNNN_obtenirListeSuivante(
                        noeudCourant);
76          }
77
78          noeudCourant = e2.lesElements;
79          while (!LCDNNN_estVide(noeudCourant)){
80                  EDNNN_ajouter(&unionEnsemble,
                        LCDNNN_obtenirElement(
                        noeudCourant));
```

```
81                  noeudCourant =
                        LCDNNN_obtenirListeSuivante(
                        noeudCourant);
82      }
83
84      return unionEnsemble;
85  }
86
87
88  EDNNN_EnsembleDeNNN EDNNN_intersection(
        EDNNN_EnsembleDeNNN e1, EDNNN_EnsembleDeNNN e2)
        {
89          EDNNN_EnsembleDeNNN intersectionEnsemble =
                EDNNN_ensemble();
90
91          LCDNNN_ListeChaineeDeNNN noeudCourant = e1
                .lesElements;
92          while (!LCDNNN_estVide(noeudCourant)) {
93                  unsigned int element =
                        LCDNNN_obtenirElement(
                        noeudCourant);
94                  if (EDNNN_estPresent(e2, element))
                         {
95                          EDNNN_ajouter(&
                                intersectionEnsemble,
                                element);
96                  }
97                  noeudCourant =
                        LCDNNN_obtenirListeSuivante(
                        noeudCourant);
98          }
99
100         return intersectionEnsemble;
101 }
102
103
104 EDNNN_EnsembleDeNNN EDNNN_soustraction(
        EDNNN_EnsembleDeNNN e1, EDNNN_EnsembleDeNNN e2)
        {
105         EDNNN_EnsembleDeNNN soustractionEnsemble =
                EDNNN_ensemble();
106
107         LCDNNN_ListeChaineeDeNNN noeudCourant = e1
                .lesElements;
108         while (!LCDNNN_estVide(noeudCourant)) {
109                 unsigned int element =
                        LCDNNN_obtenirElement(
                        noeudCourant);
```

```
110                         if (!EDNNN_estPresent(e2,
                                element)) {
111                               EDNNN_ajouter(&
                                      soustractionEnsemble
                                      , element);
112                         }
113                         noeudCourant =
                                LCDNNN_obtenirListeSuivante
                                (noeudCourant);
114         }
115
116     return soustractionEnsemble;
117 }
118
119
120 unsigned int EDNNN_obtenirPremierElement(
        EDNNN_EnsembleDeNNN unEnsemble){
121   unsigned int uneValeur;
122   uneValeur = LCDNNN_obtenirElement(unEnsemble.
          lesElements);
123   return uneValeur;
124 }
```

```
1  #include <stdbool.h>
2  #include <stdio.h>
3  #include <stddef.h>
4  #include <assert.h>
5  #include <errno.h>
6  #include <stdlib.h>
7  #include "../../include/Types/EnsembleDeDirection.
       h"
8  #include "../../include/Types/Direction.h"
9  #include "../../include/Types/
       ListeChaineeDeDirection.h"
10
11 EDD_EnsembleDeDirection EDD_ensemble(void){
12   EDD_EnsembleDeDirection ensemble;
13   ensemble.lesElements = LCDD_listeVide();
14   ensemble.nbElements = 0;
15   return ensemble;
16 }
17
18
19 void EDD_ajouter(EDD_EnsembleDeDirection *
       unEnsemble, Direction direction){
20   if (!EDD_estPresent(*unEnsemble,direction)){
21     LCDD_ajouter(&unEnsemble->lesElements,
           direction);
22     unEnsemble->nbElements = unEnsemble->
```

```
        nbElements +1;
  }
}


void EDD_retirer(EDD_EnsembleDeDirection *
    unEnsemble, Direction direction){
  LCDD_ListeChaineeDeDirection *noeudCourant = &(
      unEnsemble->lesElements);
  LCDD_ListeChaineeDeDirection precedent = NULL;
  bool directionRetiree = false;

  while (!LCDD_estVide(*noeudCourant) && !
      directionRetiree) {
    if (LCDD_obtenirElement(*noeudCourant) ==
        direction) {
      if (precedent == NULL) {
        LCDD_supprimerTete(noeudCourant);
      } else {
        LCDD_fixerListeSuivante(&precedent,
            LCDD_obtenirListeSuivante(*noeudCourant
            ));
        free(*noeudCourant);
        *noeudCourant = NULL;
      }
      unEnsemble->nbElements = unEnsemble->
          nbElements - 1;
      directionRetiree = true;
    } else {
      precedent = *noeudCourant;
      noeudCourant = &(*noeudCourant)->
          listeSuivante;
    }
  }
}

bool EDD_estPresent(EDD_EnsembleDeDirection
    unEnsemble, Direction direction){
  LCDD_ListeChaineeDeDirection noeudCourant =
      unEnsemble.lesElements;
  bool estPresent = false;

  while (!LCDD_estVide(noeudCourant) && !
      estPresent) {
    if (LCDD_obtenirElement(noeudCourant) ==
        direction) {
      estPresent = true;
    }
    noeudCourant = LCDD_obtenirListeSuivante(
```

```
        noeudCourant);
59    }
60    return estPresent;
61  }
62
63
64  unsigned int EDD_cardinalite(
        EDD_EnsembleDeDirection unEnsemble){
65    return unEnsemble.nbElements;
66  }
67
68
69  EDD_EnsembleDeDirection EDD_unionEnsemble(
        EDD_EnsembleDeDirection e1,
        EDD_EnsembleDeDirection e2){
70    EDD_EnsembleDeDirection unionEnsemble =
          EDD_ensemble();
71
72    LCDD_ListeChaineeDeDirection noeudCourant = e1.
          lesElements;
73    while (!LCDD_estVide(noeudCourant)) {
74      EDD_ajouter(&unionEnsemble,
            LCDD_obtenirElement(noeudCourant));
75      noeudCourant = LCDD_obtenirListeSuivante(
            noeudCourant);
76    }
77
78    noeudCourant = e2.lesElements;
79    while (!LCDD_estVide(noeudCourant)) {
80      EDD_ajouter(&unionEnsemble,
            LCDD_obtenirElement(noeudCourant));
81      noeudCourant = LCDD_obtenirListeSuivante(
            noeudCourant);
82    }
83
84    return unionEnsemble;
85  }
86
87
88  EDD_EnsembleDeDirection EDD_intersection(
        EDD_EnsembleDeDirection e1,
        EDD_EnsembleDeDirection e2){
89    EDD_EnsembleDeDirection intersectionEnsemble =
          EDD_ensemble();
90
91    LCDD_ListeChaineeDeDirection noeudCourant = e1.
          lesElements;
92    while (!LCDD_estVide(noeudCourant)) {
93      Direction direction = LCDD_obtenirElement(
```

```
        noeudCourant);
94    if (EDD_estPresent(e2, direction)) {
95      EDD_ajouter(&intersectionEnsemble, direction
          );
96    }
97    noeudCourant = LCDD_obtenirListeSuivante(
        noeudCourant);
98  }
99
100 return intersectionEnsemble;
101 }
102
103
104 EDD_EnsembleDeDirection EDD_soustraction(
    EDD_EnsembleDeDirection e1,
    EDD_EnsembleDeDirection e2){
105   EDD_EnsembleDeDirection soustractionEnsemble =
        EDD_ensemble();
106
107   LCDD_ListeChaineeDeDirection noeudCourant = e1.
        lesElements;
108   while (!LCDD_estVide(noeudCourant)) {
109     Direction direction = LCDD_obtenirElement(
          noeudCourant);
110     if (!EDD_estPresent(e2, direction)) {
111       EDD_ajouter(&soustractionEnsemble, direction
            );
112     }
113     noeudCourant = LCDD_obtenirListeSuivante(
          noeudCourant);
114   }
115
116   return soustractionEnsemble;
117 }
118
119
120 Direction EDD_obtenirPremierElement(
    EDD_EnsembleDeDirection unEnsemble){
121   Direction uneDirection;
122   uneDirection = LCDD_obtenirElement(unEnsemble.
        lesElements);
123   return uneDirection;
124 }
```

```
1  #include <stdbool.h>
2  #include <stddef.h>
3  #include <assert.h>
4  #include <errno.h>
5  #include <stdlib.h>
```

```
6  #include "../../include/Types/EnsembleDePileDeNNN.
       h"
7  #include "../../include/Types/
       ListeChaineeDePileDeNNN.h"
8
9
10 EDPDNNN_EnsembleDePileDeNNN EDPDNNN_ensemble(){
11         EDPDNNN_EnsembleDePileDeNNN ensemble;
12         ensemble.lesElements = LCDPDNNN_listeVide
               ();
13         ensemble.nbElements = 0;
14         return ensemble;
15 }
16
17
18 void EDPDNNN_ajouter(EDPDNNN_EnsembleDePileDeNNN *
       unEnsemble, PDNNN_PileDeNNN element){
19         if (!EDPDNNN_estPresent(*unEnsemble,
               element)) {
20           LCDPDNNN_ajouter(&(unEnsemble->
                 lesElements), element);
21         unEnsemble->nbElements++;
22     }
23 }
24
25
26 void EDPDNNN_retirer(EDPDNNN_EnsembleDePileDeNNN *
       unEnsemble, PDNNN_PileDeNNN element){
27         LCDPDNNN_ListeChaineeDePileDeNNN *
               noeudCourant = &(unEnsemble->
               lesElements);
28         LCDPDNNN_ListeChaineeDePileDeNNN precedent
               = NULL;
29         bool elementRetire = false;
30
31         while (!LCDPDNNN_estVide(*noeudCourant) &&
               !elementRetire) {
32                 if (LCDPDNNN_obtenirElement(*
                       noeudCourant) == element){
33                         if (precedent == NULL) {
34                                 LCDPDNNN_supprimerTete
                                       (noeudCourant);
35                         } else {
36                                 LCDPDNNN_fixerListeSuivante
                                       (&precedent,
                                       LCDPDNNN_obtenirListeSuivante
                                       (*noeudCourant)
                                       );
37                                 free(*noeudCourant
```

```
                                        );
38                                    *noeudCourant =
                                        NULL;
39                          }
40                          unEnsemble->nbElements--;
41                          elementRetire = true;
42                          } else {
43                          precedent = *noeudCourant;
44                          noeudCourant = &(*
                                noeudCourant)->
                                listeSuivante;
45                          }
46                  }
47  }
48
49
50  bool EDPDNNN_estPresent(
        EDPDNNN_EnsembleDePileDeNNN unEnsemble,
        PDNNN_PileDeNNN element){
51    EDPDNNN_EnsembleDePileDeNNN copie =
          EDPDNNN_copier(unEnsemble);
52    bool estPresent = false;
53    PDNNN_PileDeNNN temp;
54
55    while ((!estPresent) && (!(EDPDNNN_cardinalite(
          copie) == 0))) {
56      temp = EDPDNNN_obtenirPremierElement(copie);
57      EDPDNNN_retirer(&copie, temp);
58      if (PDNNN_egale(temp, element)) {
59        estPresent = true;
60      }
61    }
62    return estPresent;
63  }
64
65
66  unsigned int EDPDNNN_cardinalite(
        EDPDNNN_EnsembleDePileDeNNN unEnsemble){
67      return unEnsemble.nbElements;
68  }
69
70
71  EDPDNNN_EnsembleDePileDeNNN EDPDNNN_unionEnsemble(
        EDPDNNN_EnsembleDePileDeNNN e1,
        EDPDNNN_EnsembleDePileDeNNN e2){
72          EDPDNNN_EnsembleDePileDeNNN unionEnsemble
              = EDPDNNN_ensemble();
73
74          LCDPDNNN_ListeChaineeDePileDeNNN
```

```
            noeudCourant = e1.lesElements;
        while (!LCDPDNNN_estVide(noeudCourant)) {
                EDPDNNN_ajouter(&unionEnsemble,
                    LCDPDNNN_obtenirElement(
                    noeudCourant));
                noeudCourant =
                    LCDPDNNN_obtenirListeSuivante(
                    noeudCourant);
        }

        noeudCourant = e2.lesElements;
        while (!LCDPDNNN_estVide(noeudCourant)) {
                EDPDNNN_ajouter(&unionEnsemble,
                    LCDPDNNN_obtenirElement(
                    noeudCourant));
                noeudCourant =
                    LCDPDNNN_obtenirListeSuivante(
                    noeudCourant);
        }

        return unionEnsemble;
}


EDPDNNN_EnsembleDePileDeNNN EDPDNNN_intersection(
    EDPDNNN_EnsembleDePileDeNNN e1,
    EDPDNNN_EnsembleDePileDeNNN e2){
        EDPDNNN_EnsembleDePileDeNNN
            intersectionEnsemble = EDPDNNN_ensemble
            ();

        LCDPDNNN_ListeChaineeDePileDeNNN
            noeudCourant = e1.lesElements;
        while (!LCDPDNNN_estVide(noeudCourant)) {
                PDNNN_PileDeNNN element =
                    LCDPDNNN_obtenirElement(
                    noeudCourant);
                if (EDPDNNN_estPresent(e2, element
                    )) {
                        EDPDNNN_ajouter(&
                            intersectionEnsemble,
                            element);
                }
                noeudCourant =
                    LCDPDNNN_obtenirListeSuivante(
                    noeudCourant);
        }

        return intersectionEnsemble;
```

```
103 }
104
105
106 EDPDNNN_EnsembleDePileDeNNN EDPDNNN_soustraction(
        EDPDNNN_EnsembleDePileDeNNN e1,
        EDPDNNN_EnsembleDePileDeNNN e2){
107         EDPDNNN_EnsembleDePileDeNNN
                soustractionEnsemble = EDPDNNN_ensemble
                ();
108
109         LCDPDNNN_ListeChaineeDePileDeNNN
                noeudCourant = e1.lesElements;
110         while (!LCDPDNNN_estVide(noeudCourant)) {
111                 PDNNN_PileDeNNN element =
                        LCDPDNNN_obtenirElement(
                        noeudCourant);
112                 if (!EDPDNNN_estPresent(e2,
                        element)) {
113                         EDPDNNN_ajouter(&
                                soustractionEnsemble,
                                element);
114                 }
115                 noeudCourant =
                        LCDPDNNN_obtenirListeSuivante(
                        noeudCourant);
116         }
117
118         return soustractionEnsemble;
119 }
120
121
122 PDNNN_PileDeNNN EDPDNNN_obtenirPremierElement(
        EDPDNNN_EnsembleDePileDeNNN unEnsemble){
123   PDNNN_PileDeNNN unePile;
124   unePile = LCDPDNNN_obtenirElement(unEnsemble.
        lesElements);
125   return unePile;
126 }
127
128
129 EDPDNNN_EnsembleDePileDeNNN EDPDNNN_copier(
        EDPDNNN_EnsembleDePileDeNNN unEnsemble) {
130   EDPDNNN_EnsembleDePileDeNNN copie =
        EDPDNNN_ensemble();
131   LCDPDNNN_ListeChaineeDePileDeNNN temp;
132   unsigned int i = 0;
133
134   copie.lesElements = LCDPDNNN_copier(unEnsemble.
        lesElements);
```

```
135   temp = copie.lesElements;
136   while (!LCDPDNNN_estVide(temp)) {
137     i++;
138     temp = LCDPDNNN_obtenirListeSuivante(temp);
139   }
140   copie.nbElements = i;
141   return copie;
142 }
```

Les autres types

```
1  #include <stdbool.h>
2  #include <stddef.h>
3  #include <assert.h>
4  #include <errno.h>
5  #include <stdlib.h>
6  #include "../../include/Types/PileDeNNN.h"
7  #include "../../include/Types/ListeChaineeDeNNN.h"
8
9  PDNNN_PileDeNNN PDNNN_pile(){
10   return LCDNNN_listeVide();
11 }
12
13 bool PDNNN_estVide(PDNNN_PileDeNNN pile){
14   return LCDNNN_estVide(pile);
15 }
16
17
18 void PDNNN_empiler(PDNNN_PileDeNNN *pile, unsigned
       int element){
19   LCDNNN_ajouter(pile,element);
20 }
21
22
23 void PDNNN_depiler(PDNNN_PileDeNNN *pile){
24   assert(!PDNNN_estVide(*pile));
25   LCDNNN_supprimerTete(pile);
26 }
27
28
29 unsigned int PDNNN_obtenirElement(PDNNN_PileDeNNN
      pile){
30   assert(!PDNNN_estVide(pile));
31   return LCDNNN_obtenirElement(pile);
32 }
33
34 unsigned int PDNNN_longueur(PDNNN_PileDeNNN pile){
35   PDNNN_PileDeNNN temp = pile;
```

```
36    unsigned int longueur = 0;
37    while (!LCDNNN_estVide(temp)){
38      longueur++;
39      temp = LCDNNN_obtenirListeSuivante(temp);
40    }
41    return longueur;
42  }
43
44  bool PDNNN_estPresent(PDNNN_PileDeNNN pile,
        unsigned int e){
45    bool presence = false;
46    while ((!LCDNNN_estVide(pile)) && (!(presence ==
          true))){
47      if (LCDNNN_obtenirElement(pile) == e){
48        presence = true;
49      }
50      pile = LCDNNN_obtenirListeSuivante(pile);
51    }
52    return presence;
53  }
54
55
56  PDNNN_PileDeNNN PDNNN_copier(PDNNN_PileDeNNN p) {
57    return LCDNNN_copier(p);
58  }
59
60
61  bool PDNNN_egale(PDNNN_PileDeNNN p1,
        PDNNN_PileDeNNN p2) {
62    PDNNN_PileDeNNN copie1 = PDNNN_copier(p1);
63    PDNNN_PileDeNNN copie2 = PDNNN_copier(p2);
64    bool sontEgales = true;
65
66    while (sontEgales && (!PDNNN_estVide(copie1) &&
          (!PDNNN_estVide(copie2)))) {
67      if (PDNNN_obtenirElement(copie1) !=
            PDNNN_obtenirElement(copie2)) {
68        sontEgales = false;
69      }
70      PDNNN_depiler(&copie1);
71      PDNNN_depiler(&copie2);
72      if ((!PDNNN_estVide(copie1) && (PDNNN_estVide(
            copie2))) || (PDNNN_estVide(copie1) && (!
            PDNNN_estVide(copie2)))) {
73        sontEgales = false;
74      }
75    }
76    return sontEgales;
77  }
```

```c

#include <stdbool.h>
#include <stddef.h>
#include <assert.h>
#include <errno.h>
#include <stdlib.h>
#include "../../include/Types/ListeChaineeDeNNN.h"
#include "../../include/Types/MatriceDAdjacence.h"


MDA_MatriceDAdjacence MDA_creerMatriceDAdjacence(
    unsigned int nbCases){
  MDA_MatriceDAdjacence uneMatrice;
  uneMatrice.nbElements = nbCases;
  uneMatrice.tableau = NULL;
  unsigned int i;
  unsigned int j;

  uneMatrice.tableau = (unsigned int**) malloc (
      uneMatrice.nbElements * sizeof(unsigned int*)
      );
  if (!uneMatrice.tableau){
    errno = MDA_ERREUR_MEMOIRE;
  }

  for(i=1;i<=uneMatrice.nbElements;i++){
    uneMatrice.tableau[i-1] = (unsigned int*)
        malloc (uneMatrice.nbElements * sizeof(
        unsigned int));
    if (!uneMatrice.tableau[i-1]){
      errno = MDA_ERREUR_MEMOIRE;
    }

    for (j=1; j<=uneMatrice.nbElements;j++){
      uneMatrice.tableau[i-1][j-1] = 0;
    }
  }

  return uneMatrice;
}


void MDA_creerLien(MDA_MatriceDAdjacence *
    laMatrice, unsigned int case1, unsigned int
    case2){
  laMatrice->tableau[case1-1][case2-1] = 1;
  laMatrice->tableau[case2-1][case1-1] = 1;
}

```

```
43 void MDA_retirerLien(MDA_MatriceDAdjacence *
       laMatrice, unsigned int case1, unsigned int
       case2){
44   laMatrice->tableau[case1-1][case2-1] = 0;
45   laMatrice->tableau[case2-1][case1-1] = 0;
46 }
47
48 LCDNNN_ListeChaineeDeNNN MDA_obtenirLiens(
       MDA_MatriceDAdjacence laMatrice, unsigned int
       case1){
49   LCDNNN_ListeChaineeDeNNN listeDesLiens =
         LCDNNN_listeVide();
50   unsigned int i;
51
52   for (i = 0; i < laMatrice.nbElements; i++) {
53     if (laMatrice.tableau[case1-1][i] != 0) {
54       LCDNNN_ajouter(&listeDesLiens, i+1);
55     }
56   }
57   return listeDesLiens;
58 }
59
60 void MDA_libererMatrice(MDA_MatriceDAdjacence *
       laMatrice){
61   unsigned int i;
62
63   for (i = 0; i < laMatrice->nbElements; i++) {
64     free(laMatrice->tableau[i]);
65   }
66   free(laMatrice->tableau);
67   laMatrice->tableau = NULL;
68   laMatrice->nbElements = 0;
69 }
```

```
1 #include "../../include/Types/Ordre.h"
```

```
1  #include <errno.h>
2  #include <assert.h>
3  #include "../../include/Types/Direction.h"
4  #include "../../include/Types/caseEtDirection.h"
5
6
7  CD_CaseEtDirection CD_caseEtDirection(unsigned int
       c, Direction d){
8    CD_CaseEtDirection cd;
9    cd.laCase = c;
10   cd.laDirection = d;
11   return cd;
12 }
```

```
13
14
15 void CD_fixerCase(CD_CaseEtDirection* cd, unsigned
       int c){
16   (*cd).laCase = c;
17 }
18
19
20 void CD_fixerDirection(CD_CaseEtDirection* cd,
       Direction d){
21   (*cd).laDirection = d;
22 }
23
24
25 unsigned int CD_obtenirCase(CD_CaseEtDirection cd)
       {
26   return cd.laCase;
27 }
28
29
30 Direction CD_obtenirDirection(CD_CaseEtDirection
       cd){
31   return cd.laDirection;
32 }
```

```
1 #include "../../include/Types/Direction.h"
```

```
1  #include <errno.h>
2  #include <assert.h>
3  #include <stdio.h>
4  #include "../../include/Types/Direction.h"
5  #include "../../include/Types/EnsembleDeNNN.h"
6  #include "../../include/Types/EnsembleDeDirection.
       h"
7  #include "../../include/Types/MatriceDAdjacence.h"
8  #include "../../include/Types/caseEtDirection.h"
9  #include "../../include/Types/labyrinthe.h"
10 #include "../../include/Types/ListeChaineeDeNNN.h"
11
12
13 LAB_Labyrinthe LAB_labyrinthe(unsigned int
       largeurLabyrinthe, unsigned int entree,
       unsigned int sortie, Direction directionEntree,
        Direction directionSortie){
14   LAB_Labyrinthe laby;
15   CD_CaseEtDirection  lEntree;
16   CD_CaseEtDirection  laSortie;
17   MDA_MatriceDAdjacence lesLiaisons;
18
```

```
19    laby.largeur = largeurLabyrinthe;
20
21    lEntree = CD_caseEtDirection(entree,
          directionEntree);
22    laby.entree = lEntree;
23
24    laSortie = CD_caseEtDirection(sortie,
          directionSortie);
25    laby.sortie = laSortie;
26
27    lesLiaisons = MDA_creerMatriceDAdjacence((laby.
          largeur)*(laby.largeur));
28    laby.liaisonsCases = lesLiaisons;
29
30    return laby;
31  }
32
33  void LAB_casserMur(LAB_Labyrinthe* laby, unsigned
        int case1, unsigned int case2){
34    MDA_creerLien(&laby->liaisonsCases,case1,case2);
35  }
36
37  int LAB_largeur(LAB_Labyrinthe laby){
38    return laby.largeur;
39  }
40
41  CD_CaseEtDirection LAB_caseDEntree(LAB_Labyrinthe
        laby){
42    return laby.entree;
43  }
44
45  CD_CaseEtDirection LAB_caseDeSortie(LAB_Labyrinthe
        laby){
46    return laby.sortie;
47  }
48
49
50  EDD_EnsembleDeDirection LAB_directionsPossibles(
        LAB_Labyrinthe laby, unsigned int uneCase){
51    LCDNNN_ListeChaineeDeNNN laListeDesLiaisons;
52    EDD_EnsembleDeDirection directionsPossibles =
          EDD_ensemble();
53    laListeDesLiaisons = MDA_obtenirLiens(laby.
          liaisonsCases,uneCase);
54    while (!(laListeDesLiaisons==NULL)){
55      if (LCDNNN_obtenirElement(laListeDesLiaisons)
            ==(uneCase+1)){
56        EDD_ajouter(&directionsPossibles,D);
57      }
```

```
58      if (LCDNNN_obtenirElement(laListeDesLiaisons)
            ==(uneCase-1)){
59        EDD_ajouter(&directionsPossibles,G);
60      }
61      if (LCDNNN_obtenirElement(laListeDesLiaisons)
            ==((uneCase)-(laby.largeur))){
62        EDD_ajouter(&directionsPossibles,H);
63      }
64      if (LCDNNN_obtenirElement(laListeDesLiaisons)
            ==((uneCase)+(laby.largeur))){
65        EDD_ajouter(&directionsPossibles,B);
66      }
67      laListeDesLiaisons =
            LCDNNN_obtenirListeSuivante(
            laListeDesLiaisons);
68    }
69    return directionsPossibles;
70  }
71  EDNNN_EnsembleDeNNN LAB_casesAccessibles(
       LAB_Labyrinthe laby, unsigned int uneCase){
72    LCDNNN_ListeChaineeDeNNN laListeDesLiaisons;
73    EDNNN_EnsembleDeNNN casesAccessibles =
          EDNNN_ensemble();
74    laListeDesLiaisons = MDA_obtenirLiens(laby.
          liaisonsCases,uneCase);
75    while (!(LCDNNN_estVide(laListeDesLiaisons))){
76      if (LCDNNN_obtenirElement(laListeDesLiaisons)
            !=(uneCase)){
77        EDNNN_ajouter(&casesAccessibles,
              LCDNNN_obtenirElement(laListeDesLiaisons)
              );
78      }
79      laListeDesLiaisons =
            LCDNNN_obtenirListeSuivante(
            laListeDesLiaisons);
80    }
81    return casesAccessibles;
82  }
83
84  unsigned int LAB_caseDestination(LAB_Labyrinthe
       laby, unsigned int uneCase, Direction
       saDirection){
85    unsigned int caseDestination;
86    if (saDirection == H){
87      caseDestination=(uneCase-(laby.largeur));
88    }
89    if (saDirection == B){
90      caseDestination=(uneCase+(laby.largeur));
91    }
```

```
92    if (saDirection == G){
93      caseDestination=(uneCase-1);
94    }
95    if (saDirection == H){
96      caseDestination=(uneCase+1);
97    }
98    return caseDestination;
99  }
100
101
102 EDNNN_EnsembleDeNNN LAB_casesAdjacentes(
        LAB_Labyrinthe laby, unsigned int uneCase){
103   unsigned int maxl;
104   unsigned int minl;
105   unsigned int maxh;
106   unsigned int minh;
107   unsigned int i;
108   unsigned int j;
109   EDNNN_EnsembleDeNNN casesAdjacentes;
110   unsigned int caseCourante;
111
112   if ((uneCase)% (laby.largeur)==1){
113     minl=1;
114   }
115   else{
116     minl=uneCase-1;
117   }
118   if ((uneCase)%(laby.largeur)==0){
119     maxl=laby.largeur;
120   }
121   else{
122     maxl=uneCase+1;
123   }
124   if ((uneCase>=1)&&(uneCase<=laby.largeur)){
125     minh=1;
126   }
127   else{
128     minh=uneCase-1;
129   }
130   if ((uneCase>=((laby.largeur*laby.largeur)-laby.
          largeur+1))&&(uneCase<=(laby.largeur*laby.
          largeur))){
131     maxh=laby.largeur;
132   }
133   else{
134     maxh=uneCase+1;
135   }
136
137   for (i=minl;i<=maxl;i++){
```

```
138     for (j=minh-1;j<=maxh-1;j++){
139       caseCourante = i+(j*laby.largeur);
140       EDNNN_ajouter(&casesAdjacentes,caseCourante)
              ;
141     }
142   }
143   return casesAdjacentes;
144 }
145
146 EDNNN_EnsembleDeNNN LAB_casesNonAccesbiles(
      LAB_Labyrinthe laby, unsigned int uneCase){
147   return EDNNN_soustraction(LAB_casesAdjacentes(
        laby,uneCase),LAB_casesAccessibles(laby,
        uneCase));
148 }
149
150 void LAB_DirectionEntreeEtSortie(LAB_Labyrinthe
      laby, Direction* directionEntree, Direction*
      directionSortie){
151   *directionEntree = CD_obtenirDirection(laby.
        entree);
152   *directionSortie = CD_obtenirDirection(laby.
        sortie);
153 }
```