Dossier de développement logiciel

Dédale sur un ruban de Möbius

Crédit : thoiry.net

Jules Doumèche & Gwénolé Martin

2020 - Groupe 111

Sommaire

Contexte du projet :

Ce projet a été réalisé en période B (Semestre 1). Nous avons été encadrés par Mme Caraty, M. Alles-Bianchetti et M. Poitreneau.

I - Présentation du projet

1 – Objectif du projet

Dans le cadre du projet de SDA de la période B (2019-2020), nous devions coder un programme permettant, à partir d'un fichier texte contenant un labyrinthe, de trouver les « Plans du monde » situés dans le labyrinthe pour un dragon.

2 – Entrées/Sorties

Le programme est divisé en 11 fonctions (sans compter le main).

Voici un tableau récapitulatif des fonctions pour le sprint 5 avec une rapide description et les entrées/sorties (le cas échéant) :

Nom de la commande	Description	Entrée(s)/Sortie(s)
initialiserTab et detruireTab	Initialise et détruit un tableau dynamique à 3 dimensions	Lab (équivalent d'un char*** avec un booléen en plus par case)
initialiserLab et detruireLab	Initialise et détruit le labyrinthe dans le tableau	Le tableau 3D initialisé Le fichier texte du labyrinthe
initialiserDrag et detruireDrag	Initialise le dragon et réinitialise la position du Dragon dans le labyrinthe	Le labyrinthe initialisé
initialiserPos	Initialise la position des objets dans le labyrinthe (Plans du monde et Dragon)	Le labyrinthe initialisé
missionDragonSp5	Exécute l'algorithme de recherche	Le labyrinthe initialisé
estConnexe	Détermine si 2 positions sont connexes dans le labyrinthe	Le labyrinthe initialisé
mecontentement	Affiche le « mécontentement du dragon » s'il n'y a pas de solution possible	Sortie : fichier out.txt
affichersp5	Affiche le résultat de l'exécution de mission DragonSp5	Sortie : fichier out.txt

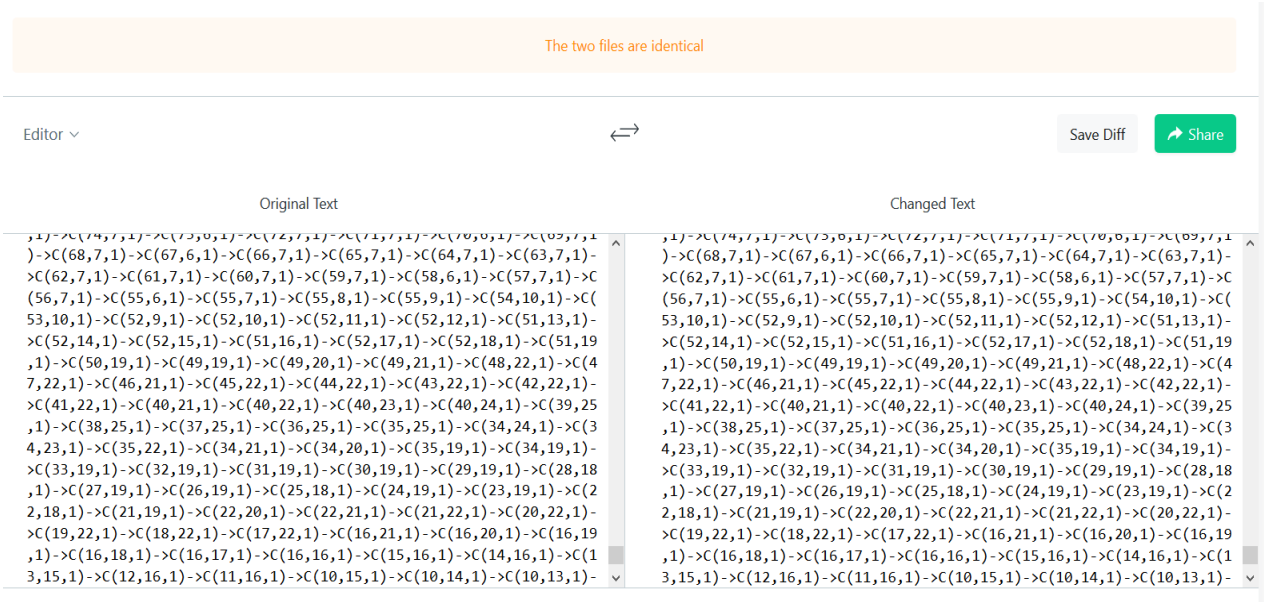
II – Bilan du développement

1 – Organisation des tests

Pour développer ce programme, nous avons utilisé une méthode de développement par sprint, avec 5 niveaux de sprints différents, nous vérifions que chaque sprint était parfaitement fonctionnel avant de passer au sprint suivant, grâce aux labyrinthes fournis (JDT) mais aussi grâce à quelques labyrinthes que nous avions faits pour l'occasion.

2 – Bilan de validation

Le sprint de plus haut niveau validé est le sprint 5 (avec le labyrinthe large). Le fichier *out* est identique au *out* de référence. Ci-dessous une capture d'écran sur DiffChecker pour labyrinthe version large du sprint 5 :



III – Bilan de projet

Le sprint 1 ne nous a pas posé de problème, les structures nécessaires ayant été vue en cours.

L'implémentation de l'algorithme de recherche pour le sprint 2 a été plus difficile, l'algorithme étant compliqué à comprendre rapidement.

Le sprint 3 n'a pas posé de problème particulier, une fois l'algorithme de recherche bien implémenté, il ne fallait que quelques modifications pour réussir ce sprint. Il fallait cependant lier les deux faces mais nous y sommes arrivés plutôt facilement.

Pour le sprint 4, la difficulté principale était la fonction estConnexe, passé cette difficulté, le reste du sprint était facile à coder.

Pour finir, le sprint 5 nécessitait juste de faire fonctionner le tout ensemble, nous y sommes arrivés sans trop de mal. Nous avons cependant dû réorganiser le code et les objets pour ne pas nous perdre. Le sprint 5 a donc principalement consisté en une réorganisation profonde de façon à le rendre le plus lisible et le plus « propre » possible.

Comme nous avons commenté les fonctions au fur et à mesure, il est normalement assez aisé de comprendre le cheminement de pensée effectué au moment de la programmation.

Nous avons pu perfectionner notre utilisation de GitHub (nous avons découvert entre le premier et le deuxième projet des fonctionnalités qui facilite encore plus la programmation en équipes), ce qui était essentiel car, à cause de la grève, il nous était plus difficile de nous réunir physiquement pour travailler sur le projet.

La principale difficulté qui ressort de ce projet était donc l'implémentation de l'algorithme, avec l'expérience du premier projet, nous avons mieux gérer notre temps et nous n'avons pas eu besoin de nous dépêcher de finir à la dernière minute.

Ce projet, bien qu'il soit plus difficile que celui que nous avions réalisé précédemment, était intéressant à réaliser. Cela permet de changer de façon de faire par rapport aux exercices de TD et de se faire une « expérience » de code. Nous avons aussi beaucoup progressé en C++, en cherchant des solutions aux innombrables petits problèmes que nous avons rencontrés en développant.

IV – Annexes

1 – Code source du sprint 5

En raison de la taille importante de cette annexe, les fonctions sont commentées à la fois dans les .h et les .cpp, pour plus faciliter la compréhension.

MAIN :

```
/**
 * @file main.cpp
 * @author Jules Doumèche, Martin Gwenole
 * @version 5 - 2019
 * @brief  Projet SDA - Sprint 5
 */

#include "tableauMb.h"
#include "dragonGame.h"
#include <clocale>

int main() {
	locale::global(locale{ "" });

	//Input stream

	ifstream in(fichier);
	if (!in) {
		cout << "Erreur!" << endl;
		cerr << "Fichier introuvable" << endl;
		return 1;
	}

	//Initialisation Labyrinthe

	Lab lab;
	in >> lab.col >> lab.lin;
	initialiserTab(lab);
	initialiserLab(in, lab);
	in.close();

	//Recherche Dragon et affichage

	Dragon drag = initialiserDrag(lab);
	missionDragonSp5(lab, drag);

	// Desallouement
	detruireDrag(drag);
	detruireLab(lab);
	detruireTab(lab);

	return 0;
}
```

Fichiers .h :

```
/**
 * @file case.h
 * @brief Définition structure Case
 * Projet SDA 1
 */
#pragma once

typedef struct {
	char car;
	bool estLu = false;
} Case;
```

```
/**
 * @file dragon.h
 * @brief Définition structure Dragon
 * Projet SDA 1
 */
#pragma once
#include "position.h"

typedef struct {
	Position pos;
} Dragon;
```

```
/**
 * @file dragonGame.h
 * @brief Initialisation du Dragon et recherche du PDM dans le labyrinthe
 * Projet SDA 1
 */
#pragma once

#include <stack>
#include "indexPositionMb.h"

/**
 * @brief Initialise le dragon dans une structure Dragon
 * @see detruireDrag, pour réinitialiser les positions du dragon en fin d'utilisation
 * @param[in-out] lab : le labyrinthe
 * @return Dragon
 */
Dragon initialiserDrag(Lab& lab);

/**
 * @brief Réinitialise à zéro les position du dragon
 * @see initialiserDrag, pour initialiser les positions du Dragon
 * @param[in-out] drag : le dragon
 * @pre drag est initialisé
```

```
 */
void detruireDrag(Dragon& drag);

/**
 * @brief Execute le sprint 5 et l'affiche
 * @see initialiserDrag, initialiserTab, initialiserLab, afficher5p5
 * @param[in-out] lab : le labyrinthe
 * @param[in-out] drag : le dragon
 * @pre Lab est initialisé, drag est initialisé
 */
void missionDragonSp5(Lab& lab, Dragon& drag);

/**
 * @brief Affiche le mécontentement du dragon si il n'y a pas de chemin
 * @see missionDragonSp5
 * @param[in] lab : le labyrinthe (constant)
 */
void mecontentement(const Lab lab);
```

```
/**
 * @file files.h
 * @brief Configuration fichier en entrée
 * Projet SDA 1
 */
#pragma once

enum { STRING_SIZE = 30 };

const char fichier[STRING_SIZE] = "in.txt";
```

```
/**
 * @file header.h
 * @brief Définition des constantes et des librairies
 * Projet SDA 1
 */
#pragma once

#include <iostream>
#include <fstream>
#include <stack>

#include "labyrinthe.h"
#include "files.h"

using namespace std;
enum { NB_DAMIERS = 2 };
```

```
/**
 * @file indexPositionMb.h
 * @brief Permet de rechercher les positions initiales du Dragon et des Plans du Monde
 * Projet SDA 1
 */
#pragma once

#include "tableauMb.h"

/**
 * @brief Initialise la position du Dragon et du PDM dans le labyrinthe
 * @see initialiserDrag
 * @param[in-out] lab : le labyrinthe
 * @param[in-out] drag : le dragon
 * @pre tab est initialisé, in est ouvert
 */
void initialiserPos(Lab& lab, Dragon& drag);

/**
 * @brief Détermine si 2 Position sont connexe dans le labyrinthe
 * @param[in] lab : le labyrinthe
 * @param[in] a : première position
 * @param[in] b : deuxième position
 * @return true si Connexe, false sinon
 */
bool estConnexe(const Lab& lab, Position a, Position b);
```

```
/**
 * @file Item.h
 * @brief Caractérisation Item en Case
 * Projet SDA 1
 */
#pragma once

#include "case.h"

typedef Case Item;
```

```
/**
 * @file labyrinthe.h
 * @brief Définition structure Lab
 * Projet SDA 1
 */
#pragma once

#include "item.h"
#include "dragon.h"

typedef struct {
	Case*** tab;
	unsigned int col;
	unsigned int lin;
	Position PDM;
} Lab;
```

```
/**
 * @file labyrintheMb.h
 * @brief Permet d'initialiser et de réinitialiser le labyrinthe, et d'afficher le
sprint 5
 * Projet SDA 1
 */
#pragma once

#include "header.h"

/**
 * @brief Initialise le labyrinthe dans un tableau 3D
 * @see detruireLab, pour réinitialiser le tableau en fin d'utilisation
 * @param[in] in : le flux d'entrée afin d'initialiser le labyrinthe
 * @param[in-out] lab : le labyrinthe à initialiser
 * @pre tab est initialisé, in est ouvert
 */
void initialiserLab(ifstream& in, Lab& lab);

/**
 * @brief Réinitialise le tableau 3D à des valeures nulles.
 * @see initialiserLab, pour initialiser le labyrinthe dans un tableau 3D
 * @param[in,out] lab : Labyrinthe contenant le tableau 3D et ses dimensions
 * @pre lab(.tab, .col, .lin) est initialisé
 */
void detruireLab(Lab& lab);

/**
 * @brief Affiche le Sprint 5
 * @see missionDragonSp5, afin d'effectuer le sprint 5 et provoquer son affichage par
l'appel de cette fonction
 * @param[in] lab : Labyrinthe contenant le tableau 3D et ses dimensions
 * @param[in] chemin : La pile contenant les coordonnées successive correspondant au
chemin réalisé par le dragon
 * @pre lab(.tab, .col, .lin) est initialisé
 */
void affichersp5(Lab& lab, stack<Position>& chemin);
```

```
/**
 * @file position.h
 * @brief Définition structure Position
 * Projet SDA 1
 */
#pragma once

typedef struct {
	unsigned int z, y, x;
} Position;
```

```
/**
 * @file tableauMb.h
 * @brief Permet d'initialiser/désallouer en mémoire le tableau 3D avant/après
affection du labyrinthe
 * Projet SDA 1
 */
#pragma once

#include "labyrintheMb.h"
```

```
/**
 * @brief Initialise le tableau 3D lab.tab allouée en mémoire dynamique
 * @see detruireTab, pour une désallocation en fin d'utilisation
 * @param[in,out] lab : structure lab, contenant la dimension du labyrinthe et le
tableau 3D associé
 */
void initialiserTab(Lab& lab);

/**
 * @brief Désalloue lab.tab
 * @see initialiserTab, pour l'initialisation du tableau
 * @param[in,out] lab : structure lab, contenant la dimension du labyrinthe et le
tableau 3D associé
 * @pre lab.tab est initialisé
 */
void detruireTab(Lab& lab);
```

Fichiers .cpp :

```
/**
 * @file dragonGame.cpp
 * @brief Initialisation du Dragon et recherche du PDM dans le labyrinthe
 * Projet SDA 1
 */
#include "dragonGame.h"

/**
 * @brief Initialise le dragon dans une structure Dragon
 * @see detruireDrag, pour réinitialiser les positions du dragon en fin d'utilisation
 * @param[in] lab : le labyrinthe
 * @return Dragon
 */
Dragon initialiserDrag(Lab& lab) {
	Dragon drag;
	initialiserPos(lab, drag);
	return drag;
}

/**
 * @brief Réinitialise à zéro les position du dragon
 * @see initialiserDrag, pour initialiser les positions du Dragon
 * @param[in-out] drag : le dragon
 * @pre drag est initialisé
 */
void detruireDrag(Dragon& drag){
	drag.pos.z = 0;
	drag.pos.y = 0;
	drag.pos.x = 0;
}

/**
 * @brief Execute le sprint 5 et l'affiche
 * @see initialiserDrag, initialiserTab, initialiserLab, afficherSp5
 * @param[in-out] lab : le labyrinthe
 * @param[in-out] drag : le dragon
 * @pre Lab est initialisé, drag est initialisé
```

```
 */
void missionDragonSp5(Lab& lab, Dragon& drag) {
	stack<Position> pile;
	stack<Position> filAriane;
	filAriane.push(drag.pos);
	pile.push(drag.pos);

	while (!(filAriane.top().z == lab.PDM.z && filAriane.top().y == lab.PDM.y &&
filAriane.top().x == lab.PDM.x) && !pile.empty()) {
		drag.pos = pile.top();
		pile.pop();

		while (!filAriane.empty() && (!estConnexe(lab, filAriane.top(),
drag.pos))) {
			filAriane.pop();
		}
		filAriane.push(drag.pos);

		if (!(drag.pos.z == lab.PDM.z && drag.pos.y == lab.PDM.y && drag.pos.x ==
lab.PDM.x)) {
			//push dans pile positions possibles de pos(i, j) de 1 à 8

			//Recherche et mouvement
			Position mouv = drag.pos;
			for (unsigned int i = 1; i <= 8; ++i) {
				mouv = drag.pos;
				switch (i) {
				case 1:
					if (drag.pos.x == 0) {
						if (lab.tab[(NB_DAMIERS - 1) -
drag.pos.z][(lab.lin - 1) - drag.pos.y][lab.col - 1].car != '#') {
							mouv.z = (NB_DAMIERS - 1) - drag.pos.z;
							mouv.y = (lab.lin - 1) - drag.pos.y;
							mouv.x = lab.col - 1;
						}
					}
					else if (lab.tab[drag.pos.z][drag.pos.y][drag.pos.x
- 1].car != '#') {
						mouv.x--;
					}
					break;
				case 2:
					if (drag.pos.y != (lab.lin - 1)) {
						if (drag.pos.x == 0) {
							if (lab.tab[(NB_DAMIERS - 1) -
drag.pos.z][lab.lin - drag.pos.y - 2][lab.col - 1].car != '#') {
								mouv.z = (NB_DAMIERS - 1) -
drag.pos.z;
								mouv.y = lab.lin - drag.pos.y -
2;
								mouv.x = lab.col - 1;
							}
						}
						else if (lab.tab[drag.pos.z][drag.pos.y +
1][drag.pos.x - 1].car != '#') {
							mouv.x--;
							mouv.y++;
						}
					}
					break;
				case 3:
```

```
                                    if (drag.pos.y != (lab.lin - 1) &&
lab.tab[drag.pos.z][drag.pos.y + 1][drag.pos.x].car != '#') {
                                          mouv.y++;
                                    }
                                    break;
                              case 4:
                                    if (drag.pos.y != (lab.lin - 1)) {
                                          if (drag.pos.x == (lab.col - 1)) {
                                                if (lab.tab[(NB_DAMIERS - 1) -
drag.pos.z][lab.lin - drag.pos.y - 2][0].car != '#') {
                                                      mouv.z = (NB_DAMIERS - 1) -
drag.pos.z;
                                                      mouv.y = lab.lin - drag.pos.y -
2;
                                                      mouv.x = 0;
                                                }
                                          }
                                          else if (lab.tab[drag.pos.z][drag.pos.y +
1][drag.pos.x + 1].car != '#') {
                                                mouv.x++;
                                                mouv.y++;
                                          }
                                    }
                                    break;
                              case 5:
                                    if (drag.pos.x == (lab.col - 1)) {
                                          if (lab.tab[(NB_DAMIERS - 1) -
drag.pos.z][(lab.lin - 1) - drag.pos.y][0].car != '#') {
                                                mouv.z = (NB_DAMIERS - 1) - drag.pos.z;
                                                mouv.y = (lab.lin - 1) - drag.pos.y;
                                                mouv.x = 0;
                                          }
                                    }
                                    else if (lab.tab[drag.pos.z][drag.pos.y][drag.pos.x
+ 1].car != '#') {
                                          mouv.x++;
                                    }
                                    break;
                              case 6:
                                    if (drag.pos.y != 0) {
                                          if (drag.pos.x == (lab.col - 1)) {
                                                if (lab.tab[(NB_DAMIERS - 1) -
drag.pos.z][lab.lin - drag.pos.y][0].car != '#') {
                                                      mouv.z = (NB_DAMIERS - 1) -
drag.pos.z;
                                                      mouv.y = lab.lin - drag.pos.y;
                                                      mouv.x = 0;
                                                }
                                          }
                                          else if (lab.tab[drag.pos.z][drag.pos.y -
1][drag.pos.x + 1].car != '#') {
                                                mouv.x++;
                                                mouv.y--;
                                          }
                                    }
                                    break;
                              case 7:
                                    if (drag.pos.y != 0 &&
lab.tab[drag.pos.z][drag.pos.y - 1][drag.pos.x].car != '#') {
                                          mouv.y--;
                                    }
                                    break;
```

```
                        case 8:
                                if (drag.pos.y != 0) {
                                        if (drag.pos.x == 0) {
                                                if (lab.tab[(NB_DAMIERS - 1) -
drag.pos.z][lab.lin - drag.pos.y][lab.col - 1].car != '#') {
                                                        mouv.z = (NB_DAMIERS - 1) -
drag.pos.z;
                                                        mouv.y = lab.lin - drag.pos.y;
                                                        mouv.x = lab.col - 1;
                                                }
                                        }
                                        else if (lab.tab[drag.pos.z][drag.pos.y -
1][drag.pos.x - 1].car != '#') {
                                                mouv.x--;
                                                mouv.y--;
                                        }
                                }
                                break;
                        default:
                                cerr << "Erreur lors de la recherche de chemin" <<
endl;
                        }
                        if (!(mouv.z == drag.pos.z && mouv.y == drag.pos.y &&
mouv.x == drag.pos.x)
                                && lab.tab[mouv.z][mouv.y][mouv.x].estLu == false) {
                                pile.push(mouv);
                        }
                }
                lab.tab[drag.pos.z][drag.pos.y][drag.pos.x].estLu = true;
        }
    }

    if (filAriane.top().z == lab.PDM.z && filAriane.top().y == lab.PDM.y &&
filAriane.top().x == lab.PDM.x) {
            unsigned int size = filAriane.size();
            stack<Position> chemin;
            chemin = filAriane;
            for (unsigned int i = 0; i < size; ++i) {

    lab.tab[filAriane.top().z][filAriane.top().y][filAriane.top().x].car = 'C';
                    filAriane.pop();
            }
            affichersp5(lab, chemin);
    }
    else {
            mecontentement(lab);
    }
}

/**
 * @brief Affiche le mécontentement du dragon si il n'y a pas de chemin
 * @see missionDragonSp5
 * @param[in] lab : le labyrinthe (constant)
 */
void mecontentement(const Lab lab) {


    cout << lab.col << " " << lab.lin << endl;
    for (unsigned int i = 0; i < NB_DAMIERS; ++i) {
            //Lire tab[i]
            for (unsigned int j = 0; j < lab.lin; ++j) {
                    //Lire ligne j
```

```
                        for (unsigned int l = 0; l < lab.col; ++l) {

                                if (lab.tab[i][j][l].car == 'P') {
                                        cout << "P";
                                        continue;
                                }

                                cout << '#';
                        }
                        cout << endl;
                }
                cout << endl;
        }
        cout << endl;

}
```

```
/**
 * @file indexPositionMb. pp
 * @brief Permet de rechercher les positions initiales du Dragon et des Plans du Monde
 * Projet SDA 1
 */
#include "indexPositionMb.h"
#include <iostream>

/**
 * @brief Initialise la position du Dragon et du PDM dans le labyrinthe
 * @see initialiserDrag
 * @param[in-out] lab : le labyrinthe
 * @param[in-out] drag : le dragon
 * @pre tab est initialisé, in est ouvert
 */
void initialiserPos(Lab& lab, Dragon& drag) {
        bool initDrag = false;
        bool initPDM = false;
        for (unsigned int i = 0; i < NB_DAMIERS; ++i) {
                        for (unsigned int j = 0; j < lab.lin; ++j) {
                                for (unsigned int l = 0; l < lab.col; ++l) {
                                        if (lab.tab[i][j][l].car == 'D') {
                                                drag.pos.z = i;
                                                drag.pos.y = j;
                                                drag.pos.x = l;
                                                initDrag = true;
                                        }
                                        if (lab.tab[i][j][l].car == 'P') {
                                                lab.PDM.z = i;
                                                lab.PDM.y = j;
                                                lab.PDM.x = l;
                                                initPDM = true;
                                        }
                                        if (initDrag && initPDM) {
                                                return;
                                        }
                                }
                        }
                }
        cerr << "Erreur! Impossible de trouver le Dragon Et/Ou les Plans du Monde" <<
endl;
        return;
}
```

```
/**
 * @brief Détermine si 2 Position sont connexe dans le labyrinthe
 * @param[in] lab : le labyrinthe
 * @param[in] a : première position
 * @param[in] b : deuxième position
 * @return true si Connexe, false sinon
 */
bool estConnexe(const Lab& lab, Position a, Position b) {
	//Orientation


	if (a.z == b.z) { // a et b sur le meme paneau
		if (a.y == b.y) { // a et b meme ligne
			if ((a.x - b.x == 1) || (b.x - a.x == 1)) {
				return true;
			}
			else
				return false;
		}
		else { // a et b pas meme ligne
			if (a.x == b.x) { // a et b meme colonne et pas meme ligne
				if ((a.y - b.y == 1) || (b.y - a.y == 1)) {
					return true;
				}
				else
					return false;
			}
			else {// a et b pas meme ligne, pas meme colonne
				if (a.x - b.x == 1) {
					if ((a.y - b.y == 1) || (b.y - a.y == 1)) {
						return true;
					}
					else
						return false;
				}
				else if (b.x - a.x == 1) {
					if ((a.y - b.y == 1) || (b.y - a.y == 1)) {
						return true;
					}
					else
						return false;
				}
				return false;
			}
		}
	}
	else {// a et b PAS sur la même face
		if (a.x == 0) {
			if (b.x != lab.col - 1) {
				return false;
			}
		}
		else if (b.x == 0) {
			if (a.x != lab.col - 1) {
				return false;
			}
			else {
				Position temp = b;
				b = a;
				a = temp;
			}
```

```
			}

			//on a a.x = 0 et b.x = dernière ligne
			if ((b.y == (lab.lin - 2 - a.y)) || (b.y == (lab.lin - 1 - a.y)) || (b.y
== (lab.lin - a.y))) {
				return true;
			}
			else
				return false;
		}
		return false;
}
```

```
/**
 * @file labyrintheMb.cpp
 * @brief Permet d'affecter et de réinitialiser le labyrinthe, et d'afficher le sprint
5
 * Projet SDA 1
 */
#include "labyrintheMb.h"

/**
 * @brief Initialise le labyrinthe dans un tableau 3D
 * @see detruireLab, pour réinitialiser le tableau en fin d'utilisation
 * @param[in] in : le flux d'entrée afin d'initialiser le labyrinthe
 * @param[in-out] lab : le labyrinthe à initialiser
 * @pre tab est initialisé, in est ouvert
 */
void initialiserLab(ifstream& in, Lab& lab) {
	for (unsigned int i = 0; i < NB_DAMIERS; ++i) {
		//Ecrit tableau[i]
		for (unsigned int j = 0; j < lab.lin; ++j) {
			//Ecrit ligne j
			for (unsigned int l = 0; l < lab.col; ++l) {
				in >> lab.tab[i][j][l].car;
			}
		}
	}
}

/**
 * @brief Réinitialise le tableau 3D à des valeures nulles.
 * @see initialiserLab, pour initialiser le labyrinthe dans un tableau 3D
 * @param[in,out] lab : Labyrinthe contenant le tableau 3D et ses dimensions
 * @pre lab(.tab, .col, .lin) est initialisé
 */
void detruireLab(Lab& lab) {
	for (unsigned int i = 0; i < NB_DAMIERS; ++i) {
		//Supprime tab[i]
		for (unsigned int j = 0; j < lab.lin; ++j) {
			//Supprime ligne j
			for (unsigned int l = 0; l < lab.col; ++l) {
				lab.tab[i][j][l].car = NULL;
				lab.tab[i][j][l].estLu = false;
			}
		}
	}
}
```

```
/**
 * @brief Affiche le Sprint 5
 * @see missionDragonSp5, afin d'effectuer le sprint 5 et provoquer son affichage par
l'appel de cette fonction
 * @param[in] lab : Labyrinthe contenant le tableau 3D et ses dimensions
 * @param[in] chemin : La pile contenant les coordonnées successives correspondant au
chemin réalisé par le dragon
 * @pre lab(.tab, .col, .lin) est initialisé
 */
void affichersp5(Lab& lab, stack<Position>& chemin) {



	cout << lab.col << " " << lab.lin << endl;
	for ( unsigned int i = 0; i < NB_DAMIERS; ++i) {
		//Lire tab[i]
		for (unsigned int j = 0; j < lab.lin; ++j) {
			//Lire ligne j
			for (unsigned int l = 0; l < lab.col; ++l) {
				cout << lab.tab[i][j][l].car;
			}
			cout << endl;
		}
		cout << endl;
	}
	unsigned int size = chemin.size();
	for (unsigned int i = 0; i < size; ++i) {
		cout << "C(" << chemin.top().x << "," << chemin.top().y << "," <<
chemin.top().z + 1 << ")->";
		chemin.pop();
	}
	cout << endl;
}
```

```
/**
 * @file tableauMb.h
 * @brief Permet d'initialiser/désallouer en mémoire le tableau 3D avant/après
affection du labyrinthe
 * Projet SDA 1
 */
#include "tableauMb.h"

/**
 * @brief Initialise le tableau 3D lab.tab allouée en mémoire dynamique
 * @see detruireTab, pour une désallocation en fin d'utilisation
 * @param[in,out] lab : structure lab, contenant la dimension du labyrinthe et le
tableau 3D associé
 */
void initialiserTab(Lab& lab) {
	lab.tab = new Case**[NB_DAMIERS];
	for (unsigned int i = 0; i < NB_DAMIERS; ++i) {
		lab.tab[i] = new Case*[lab.lin];
		for (unsigned int j = 0; j < lab.lin; ++j) {
			lab.tab[i][j] = new Case[lab.col];
		}
	}
}
```

```
/**
 * @brief Désalloue lab.tab
 * @see initialiserTab, pour l'initialisation du tableau
 * @param[in,out] lab : structure lab, contenant la dimension du labyrinthe et le
tableau 3D associé
 * @pre lab.tab est initialisé
 */
void detruireTab(Lab& lab) {
	for (unsigned int i = 0; i < NB_DAMIERS; ++i) {
		for (unsigned int j = 0; j < lab.lin; ++j) {
			delete[] lab.tab[i][j];
		}
		delete[] lab.tab[i];
	}
	delete[] lab.tab;
	lab.col = 0;
	lab.lin = 0;
}
```

2 – Trace d'exécution du sprint 5

IN (small, le large prendrait trop de place inutilement et cela ne serait pas lisible) :

```
30 4
#######D######################
+#++#+++++#####++++########++++
#+##+#+###++#######++###+##+#++
##############################

##############################
+#+#######++++++####++########+
+++++P+++++++++##+++++#+++++++++
##############################
```

OUT (le C de la 8ème ligne est décalé, mais il y a bien 30 caractères sur chaque ligne) :

```
30 4
#######C######################
C#CC#C+CC+#####++++########++++
#C##C#C###++#######++###+##+#++
##############################

##############################
+#+#######CCCCCC####CC#######C
+++++CCCCCCCCC##CCCC#CCCCCCCCC
##############################
```

C(5,2,2)->C(6,2,2)->C(7,2,2)->C(8,2,2)->C(9,2,2)->C(10,2,2)->C(10,1,2)->C(11,2,2)->C(11,1,2)->C(12,2,2)->C(13,1,2)->C(12,1,2)->C(13,2,2)->C(14,1,2)->C(15,1,2)->C(16,2,2)->C(17,2,2)->C(18,2,2)->C(19,2,2)->C(20,1,2)->C(21,2,2)->C(21,1,2)->C(22,2,2)->C(23,2,2)->C(24,2,2)->C(25,2,2)->C(26,2,2)->C(27,2,2)->C(28,2,2)->C(29,2,2)->C(29,1,2)->C(0,1,1)->C(1,2,1)->C(2,1,1)->C(3,1,1)->C(4,2,1)->C(5,1,1)->C(6,2,1)->C(7,1,1)->C(8,1,1)->C(7,0,1)->