



UNIVERSITÉ  
DE LORRAINE



Saint-Dié-des-Vosges

# CONCEPTION & SIMULATION D'ALGORITHMES POUR ROBOT TONDEUSE



AISSI AYOUB  
&  
DELANDHUY MATTEO

DUT INFO2 - 2018/2019

TUTEUR : A.BOURJIJ

## **SOMMAIRE**

I. Introduction.....	3
II. Démarche de travail.....	3
III. Résultats de l'application.....	5
1. Console.....	5
a) Initialisation du terrain .....	6
b) Algorithme simple .....	7
c) Algorithme spirale .....	9
d) Algorithme Random .....	9
e) Les méthodes de calculs de distance et du temps.....	10
f) Fonctionnement de la tonte du terrain par le robot .....	11
g) Résultat final .....	11
h) Création de la fonction rectangleATondre .....	12
2. Interface graphique.....	16
a) Initialisation des images du terrain.....	17
b) L'algorithme Spirale .....	18
c) L'algorithme Normal .....	19
d) Algorithme Random .....	20
IV Bilan .....	21
IV. Perspectives.....	22
V. ANNEXES.....	23
1. Planning.....	23
2. Diagrammes UML.....	24
3. Notice du main.....	25
a) Partie console .....	25
b) Partie graphique .....	27
4. Documentation JAVADOC.....	28
5. Cahier des charges.....	28
6. Codes sources.....	29
7. Rapport en anglais.....	29

# **I. Introduction**

Couper l'herbe, tout a commencer à la faux, il fallait beaucoup de temps. Ensuite il y a eu les tondeuses, bémol : elles demandent encore un effort humain, font beaucoup de bruit et prennent du temps. Il y a eu les tracteurs tondeuses, belle amélioration mais demande encore du temps et n'est pas très précis, corvée lassante. Pourquoi ne pas combiner le meilleur de tous ces éléments ? J'ai nommé la robotique au service de la coupe de l'herbe : le robot tondeuse.

Avant la conception du robot en lui-même, il faut développer son logiciel. Notre rôle a été de développer des algorithmes qui permettent au robot de se déplacer sur une surface, ainsi que simuler le déroulement de ces algorithmes sur une console et une interface graphique.

# **II. Démarche de travail**

Pour commencer ce projet, nous avons analysé ce qui existait déjà. Plusieurs marques de robot tondeuse sont sur le marché. Les technologies utilisées varient selon la gamme de prix.

Nous avons interrogé Abdelattif Bourjij notre tuteur sur le fonctionnement du robot tondeuse qu'il possède.

Nous avons fixé les besoins du plus important au moins important, ainsi que estimer un temps pour les développer. Afin de bien nous organiser, nous avons créé un diagramme de GANTT disponible en annexe.

Nous avons dessiné un diagramme UML pour organiser la disposition des classes, les fonctions, ainsi il est plus facile de voir ce qu'il y a à faire. En réalité, au fur et à mesure que l'on a développé le code nous avons rencontré des problèmes qui nous ont obligés à créer d'autres fonctions, à adapter des types de données, à créer de nouvelles classes.

Nous avons réfléchi aux différents algorithmes de déplacement qu'il existait. Nous avons d'abord pensé aux algorithmes qui conviennent pour un terrain sans obstacles. Nous avons donc développé l'algorithme simple, c'est un algorithme qui permet au robot de se déplacer ligne par ligne sur le terrain, une fois arrivé à la limite du terrain, il descend d'une case et continue sur la ligne soit à droite soit à gauche en fonction de sa position.

Cet algorithme est efficace, néanmoins, il serait difficile et parfois impossible que le robot évite des obstacles. De plus le robot doit soit connaître les dimensions du terrain à l'avance soit posséder un capteur qui valide si oui ou non le robot peut avancer (il ne peut pas avancer lorsque il y a un signal électrique ou bien si il y a un obstacle).

Nous avons également pensé à l'algorithme en spirale. Il part fonctionne simplement de la sorte :

→→→→→↓↓  
↑→→→→↓↓  
↑↑←←←←↓  
↑←←←←←←

Cette algorithme est efficace mais il existe les mêmes problèmes qu'avant, dans le cas d'obstacles, cette algorithme n'est pas adapté, de plus on peut imaginer que en situation réelle le robot se décalerait petit à petit, de plus les roues peuvent patiner et décaler la localisation dont le robot pense être. On peut contrer cela en équipant le robot d'un GPS ultra précis.

Enfin nous avons développé l'algorithme qui équipent la plupart des robots tondeuse ou d'aspiration pour maison ou de piscine, c'est l'algorithme random.

Son principe est celui-ci :

Le robot part dans une direction au hasard, lorsqu'il rencontre un obstacle ou une limite de terrain, il repart dans un angle au hasard, de cette sorte le robot peut évoluer à travers le terrain sans le connaître ni posséder de puce gps.

Pourquoi un angle au hasard ?

Afin de s'adapter à tout terrain et à tout obstacle.

Pour développer l'application, nous avons utilisé le langage de programmation JAVA sur Eclipse.

A chaque avancement du projet, nous avons uploadé notre travail sur github à cette adresse :

<https://github.com/matteodelandhuy/PT3-Robot-tondeuse>

Pour commencer à développer du code, il existait 2 possibilités, soit utiliser la librairie TURTLE de java consultable à cette adresse :

<http://sites.asmsa.org/java-turtle/documentation>

Soit nous pouvions choisir de refaire toutes les fonctions par nous même, c'est la solution que nous avons choisi afin de nous améliorer dans le développement d'algorithmes, mais aussi pour faciliter la migration de cette application sur une autre plateforme par exemple sur un robot avec Arduino.

Le travail a donc été séparé en 2. D'un côté, nous avons développé le code pour avoir un résultat sur console. De l'autre côté il fallait implémenté le code pour avoir un résultat graphique.

Pour représenter les différents éléments, ce sont des chiffres qui sont associés à l'image du robot, de sa base, de l'herbe, de l'herbe tondu et des obstacles.

Une fois la partie console terminé, un soucis se posait. En affichant chaque étape de l'avancement de l'algorithme random, le temps passé était très long , d'autant plus si le terrain était grand.

Le soucis de l'algorithme random est que tondre le terrain à 100 % est très long car il existe toujours une case où le robot met du temps à passer dessus par hasard. Fort de ce constat, nous avons développé des fonctions qui ont pour rôle de calculer la distance parcourue ainsi que le temps mis par le robot pour tondre 50 %,70 %, 90 %, 100 % du terrain. Cela peut permettre également d'arrêter la simulation à 50 % du terrain si on le souhaite.

Afin d'avoir un résultat plus rapide de la simulation, nous avons mis en place une fonction qui permet d'afficher ou non chaque étape de déplacement du robot car ce qui consomme beaucoup de temps est l'affichage.

Ainsi les déplacements se font sans s'afficher, et l'application ne fournit très rapidement le résultat final.

### **III. Résultats de l'application**

#### **1. Console**

Comme dit précédemment, les éléments du terrain sont représentés par des chiffres. Cela vient de la contrainte que la console ne peut afficher directement des images.

Voici la signification des nombres :

9 : herbe non coupé

1 : herbe coupé

-5 : robot

3 : obstacles

-3 : base de recharge du robot

### a) Initialisation du terrain

```
999999
999999
999999
999999
999999
999999
999999

    public void initialiser() {
        for (int i = 0; i < y+1; i++) {
            for (int j = 0; j < x+1 ; j++) {
                g[i][j] = hauteurHerbe;
            }
        }
    }
```

Voici la fonction qui est appelé à chaque fois qu'il y a besoin d'afficher le terrain :

```
    public void afficher(int[][] g) {

        if(afficherTerrain == true) {
            for (int l=0; l<y+1 ; l++) {
                for (int c=0; c<x+1 ; c++) {
                    System.out.print(g[l][c]);
                }
            }
        }
    }
```

→ On parcourt le terrain et on affiche chaque case, les valeurs des cases sont changés selon si la pelouse est tondu ou non ,si il se trouve le robot ou autre chose.

## b) Algorithme simple

-599999	1-599999	11-5999	111-599	1111-59	11111-5	111111	111111
999999	999999	999999	999999	999999	999999	99999-5	9999-51
999999	999999	999999	999999	999999	999999	999999	999999
999999	999999	999999	999999	999999	999999	999999	999999
999999	999999	999999	999999	999999	999999	999999	999999
999999	999999	999999	999999	999999	999999	999999	999999

Comme prévu le robot évolue de cette manière :



J'ai prévu de placer la base du robot avec cette fonction :

```
public int[][] placerRobotEtBase(int[][] g) {  
    g[y][x] = IMGROBOT;  
    g[yLocalisationBase][xLocalisationBase] = IMGBASE;  
    return g;  
}
```

→ Comme on peut le voir, c'est le terrain qui est mis à jour car c'est le terrain qu'on retourne.

→ Le terrain est un tableau à double entrée.

Voici la fonction qui permet au robot d'éviter la base avec cet algorithme :

```
/**contourner la base*/  
public void EviterBase(int[][] g, int y,int x) {  
    if (xLocalisationBase == 0) {  
        if (yLocalisationBase % 2 == 0) {  
            D();  
            B();  
            boolB = true;  
        }  
        if (yLocalisationBase % 2 == 1) {  
            G();  
            B();  
            boolB = true;  
        }  
    }  
    if (xLocalisationBase == t.getX()) {  
        if (yLocalisationBase % 2 == 0) {  
            B();  
            D();  
            boolB = true;  
        }  
        if (yLocalisationBase % 2 == 1) {  
            G();  
            B();  
            boolB = true;  
        }  
    }  
    // si base sur ligne paire  
    if (yLocalisationBase % 2 == 0) {  
        B();  
        D();  
        D();  
        H();  
    }  
}
```



```

// si base sur ligne impaire
if (yLocalisationBase % 2 == 1) {
    B();
    G();
    G();
    H();
}
}

```

→ Ici se dégage déjà la complexité des différentes situations qu'il peut y avoir  
→ Il est question ici d'agir selon le placement de l'obstacle par rapport au robot.

Les fonctions B,G,H,D sont des méthodes qui permettent au robot de se déplacer de 1 case selon la direction.

### c) Algorithme spirale

```

111111
999991
999991
-599991
199991
111111

```

```

111111
999991
-599991
199991|
199991
111111

```

```

111111
-599991
199991
199991
199991
111111

```

```

111111
1-59991
199991
199991
199991
111111

```

→ Comme prévu l'algorithme suit un chemin en spirale de l'extérieur vers l'intérieur

→ L'algorithme s'arrête lorsque le robot a fait le même nombre de déplacement qu'il existe de case.

La fonction TerrainTendu() dans la classe Terrain permet de comptabiliser le nombre de case tondu.

### d) Algorithme Random

```

1111111111111111
1111111111111111
1133333331111111
1133333331111111
1133333331111111
1133333331111111
1133333331111111
1133333331111111
1133333331111111
1133333331111111
1111111111111111
1111111111111111
111-511111111111
1111111111111111
1111111111191111
1111111111111111

```

```

1111111111111111
1111111111111111
1133333331111111
1133333331111111
1133333331111111
1133333331111111
1133333331111111
1133333331111111
1133333331111111
1133333331111111
1111111111111111
1111111111111111
1111111111111111
11-511111111111
1111111111191111
1111111111111111

```

```

1111111111111111
1111111111111111
1133333331111111
1133333331111111
1133333331111111
1133333331111111
1133333331111111
1133333331111111
1133333331111111
1133333331111111
1111111111111111
1111111111111111
1111111111111111
1111111111111111
11-51111111191111
1111111111111111

```

légende :

1 : herbe tondu

9 : herbe non tondu

3 : obstacle

-5 : robot

→ Comme l'on peut voir, le robot s'est déplacé ici vers la diagonale bas gauche avec la méthode BG() de la classe Robot.

Lors du développement de l'algorithme Random, il s'est posé un problème.

Le problème est que le robot ne peut partir avec un angle au hasard sur le terrain car le terrain est un tableau.

Nous avons donc trouvé cette solution :

Lorsque le robot est arrêté par un obstacle ou qu'il arrive au bord du terrain :

1<sup>er</sup> étape : une direction est choisi au hasard

NombreAleatoire() de la classe AlgorithmeRandom permet de générer un nombre qui décidera de la direction. Chaque nombre est associé de base à une direction.

NombreAleatoireDirection() de la classe AlgorithmeRandom permet de générer un nombre qui sera le nombre de déplacements que le robot fera.

Si la direction à prendre n'est pas possible (vérifier par la fonction CasePossible() de la classe Terrain), une nouvelle direction est choisie au hasard.

### **e) Les méthodes de calculs de distance et du temps**

Le calcul du temps et de la vitesse est lié à la formule  $v = d/t$  , soit vitesse = distance/ temps.

La vitesse du robot est demandé à l'utilisateur lors de l'initialisation des paramètres du robot. A partir de ça, la distance parcourue et le temps passé par le robot est incrémenté à chacun de ses déplacements.

Exemple :

```
public double convertDistancetoTemps() {  
    double t = distanceEnMetre() / r.getVitesse();  
    return t ;  
}
```

## f) Fonctionnement de la tonte du terrain par le robot

Tout se passe dans la fonction qui permet au robot de se déplacer vers une direction.

Prenons l'exemple de la fonction qui permet au robot de faire un déplacement de 1 case vers la droite.

Contenu de la fonction D() :

```
public void D() {
    if (caseTondable(g, y, x+1) == true) {
        x++;
        g = t.getG();
        g[y][x]=IMGROBOT;
        distanceUpdate();
        tondre(g, y, x-1);
        t.afficher(g);
    }
    else {
        casePossible = false;
        System.out.println("La case suivante n'est pas
tondable");
    }
}
```

→ On vérifie si la case suivante est tondable, une case n'est tondable que si il y a de l'herbe non tondu ou tondu, de cette manière on sait si on peut se déplacer sur la case suivante.

→ Si on peut aller sur la case suivante on y va et la case prends comme valeur le robot.

→ On tonds la case précédente.

Conclusion : L'herbe de la case est tondu qu'une fois que le robot est sur la case suivante.

## g) Résultat final

Que l'on active ou non l'affichage du déplacement du robot visible ou non le résultat final de l'application est la distance que le robot a parcouru et le temps qu'il a mis afin de tondre 50%, 70 %, 90 % et 100 % du terrain.

```

Le robot a tondu 50 % du terrain en 0.702 kms.
0heures, 29 minutes, 14 secondes.
Le robot a tondu 75 % du terrain en 2.259 kms.
1heures, 34 minutes, 7 secondes.
Le robot a tondu 90 % du terrain en 11.007 kms.
7heures, 38 minutes, 37 secondes.
Le robot a tondu 100 % du terrain en 11.013 kms.
7heures, 38 minutes, 52 secondes.

```

→ Le calcul pour la tonte à 100 % du terrain semble erroné, c'est un point qu'il faudra corriger.

Conclusion : Le résultat final réponds bien à notre problématique de simulation qui avait pour but de connaître en combien de temps un robot sur un terrain donné mettra en temps pour tondre différents pourcentages du terrain.

Point négatif : Lorsque l'utilisateur rentre un très grand terrain et utilise l'algorithme random, l'application peut mettre beaucoup de temps à atteindre les 100 % de surface tondue, de ce fait il est plus raisonnable de paramétrer l'algorithme pour qu'il s'arrête à 50 % ou 75 % du terrain.

## h) Création de la fonction rectangleATondre

Cette fonction consiste à analyser le terrain et les obstacles qu'ils possèdent et à partir de là de répertorier les coordonnées de tous les rectangles que l'on peut tondre sans rencontrer d'obstacles. Cela est bien pratique car si le robot connaît directement les parties à tondre, avec l'algorithme A\* on peut le faire aller sur chaque rectangle et couper l'herbe soit avec l'algorithme simple ou le spirale.

Le but final de cette méthode est de pouvoir s'adapter à toutes les configurations de terrain et d'obstacles et d'être bien plus rapide que l'algorithme Random, cela se faisant en ne passant uniquement ou presque sur les zones d'herbe non encore tondue.

Exemple :

9 : herbe

-5 : robot

3 : obstacle

```

-5999999999999999
9999999999999999
9933339999999999
9933339999999999
9933339999999999
9933339999999999
9999999999999999
9999999999999999
9999999999999999
9999999999999999
9999999999999999
9999999999999999

```

Le terrain est maintenant décomposé en 5 rectangles d'herbe sans obstacles.

Pourquoi utiliser l'algorithme A\* ?

Afin que le robot puisse aller à chaque début de parcelle d'herbe sans être arrêté par un obstacle tel qu'il pourrait se passer avec un algorithme glouton.

Explication du code :

Premièrement, nous allons répertorier tous les bouts de lignes qui sont sans obstacles. Je stocke temporairement une ligne dans `ligneATondre[]`. On aura `ligneATondre[0]` = x début de ligne, `ligneATondre[1]` = y début de ligne, `ligneATondre[2]` = x fin de ligne, `ligneATondre[3]` = y fin de ligne. Si l'algo rencontre un obstacle, on considère que la fin de la ligne d'herbe s'arrête une case avant l'obstacle. Toutes les lignes seront stockées dans `listeLigneATondre[i][j]`. On a `i` = n-ième élément du tableau et `j` qui peut prendre la valeur 0,1,2,3 qui possède les données de `ligneATondre` avec ses 4 valeurs.

Une fois que tout est répertorié il faut maintenant créer des rectangles à partir des lignes, pour cela il faudra faire le test de si une ligne possède le même x début et le même x de fin soit `listeLigneATondre[i][0]` et `listeLigneATondre[i][2]`, mais il faudra également que le coordonné y de la ligne soit à maximum 1 du y de la ligne comparé ce qui nous donne :

```
while(listeLigneATondre[p][1] + h == listeLigneATondre[m+1][1]-1 &&  
listeLigneATondre[p][3] + h == listeLigneATondre[m+1][3]-1) {
```

La variable `h` nous permet de pouvoir l'incrémenter à chaque tour dans la boucle et donc de pouvoir comparer sous condition qu'ils aient les même coordonnées x les lignes entre elle, c'est à dire comparer la ligne 1 à la 2 puis la 3 à la 4 puis la 5 à la 6 etc.

Malheureusement `listeLigneATondre` ne possède pas dans l'ordre les lignes à tondre selon leur y, en effet sur une même ligne du terrain de coordonnée y on peut avoir lignes à tondre si un obstacle est entre les 2. De ce fait il faut comparer toute la `listeLigneATondre` entre elle pour voir si des lignes peuvent être mise en commun.

Néanmoins, on observe que pour comparer tous les éléments du tableaux entre eux à chaque nous oblige à ajouter une boucle en plus (elle est mise en commentaire dans le code). Le bémol de cette boucle est que elle ajoute une grande complexité de calcul au code et donc les résultats ne sont plus immédiat comme avant, voilà pourquoi ils sont mis en commentaire. Donc en retirant cette boucle, on obtient juste des rectangles de 2 lignes avec un coordonnée y différent maximum du terrain.

Voici son résultat :

listeLigneATondre :

```
rang: 0 rang: 0 résultats: 0
rang: 0 rang: 1 résultats: 0
rang: 0 rang: 2 résultats: 14
rang: 0 rang: 3 résultats: 0
rang: 1 rang: 0 résultats: 0
rang: 1 rang: 1 résultats: 1
rang: 1 rang: 2 résultats: 14
rang: 1 rang: 3 résultats: 1
rang: 2 rang: 0 résultats: 0
rang: 2 rang: 1 résultats: 2
rang: 2 rang: 2 résultats: 1
rang: 2 rang: 3 résultats: 2
rang: 3 rang: 0 résultats: 6
rang: 3 rang: 1 résultats: 2
rang: 3 rang: 2 résultats: 14
rang: 3 rang: 3 résultats: 2
rang: 4 rang: 0 résultats: 0
rang: 4 rang: 1 résultats: 3
rang: 4 rang: 2 résultats: 1
rang: 4 rang: 3 résultats: 3
rang: 5 rang: 0 résultats: 6
rang: 5 rang: 1 résultats: 3
rang: 5 rang: 2 résultats: 14
rang: 5 rang: 3 résultats: 3
rang: 6 rang: 0 résultats: 0
rang: 6 rang: 1 résultats: 4
rang: 6 rang: 2 résultats: 1
rang: 6 rang: 3 résultats: 4
rang: 7 rang: 0 résultats: 6
rang: 7 rang: 1 résultats: 4
rang: 7 rang: 2 résultats: 14
rang: 7 rang: 3 résultats: 4
rang: 8 rang: 0 résultats: 0
rang: 8 rang: 1 résultats: 5
rang: 8 rang: 2 résultats: 1
rang: 8 rang: 3 résultats: 5
rang: 9 rang: 0 résultats: 6
rang: 9 rang: 1 résultats: 5
rang: 9 rang: 2 résultats: 14
rang: 9 rang: 3 résultats: 5
rang: 10 rang: 0 résultats: 0
rang: 10 rang: 1 résultats: 6
rang: 10 rang: 2 résultats: 14
rang: 10 rang: 3 résultats: 6
rang: 11 rang: 0 résultats: 0
rang: 11 rang: 1 résultats: 7
rang: 11 rang: 2 résultats: 14
rang: 11 rang: 3 résultats: 7
rang: 12 rang: 0 résultats: 0
rang: 12 rang: 1 résultats: 8
rang: 12 rang: 2 résultats: 14
rang: 12 rang: 3 résultats: 8
rang: 13 rang: 0 résultats: 0
rang: 13 rang: 1 résultats: 9
rang: 13 rang: 2 résultats: 14
rang: 13 rang: 3 résultats: 9
rang: 14 rang: 0 résultats: 0
rang: 14 rang: 1 résultats: 10
rang: 14 rang: 2 résultats: 14
rang: 14 rang: 3 résultats: 10
```

listeRectangleATondre :

```
rang: 0 rang: 0 résultats: 0
rang: 0 rang: 1 résultats: 0
rang: 0 rang: 2 résultats: 14
rang: 0 rang: 3 résultats: 1
rang: 1 rang: 0 résultats: 0
rang: 1 rang: 1 résultats: 1
rang: 1 rang: 2 résultats: 14
rang: 1 rang: 3 résultats: 1
rang: 2 rang: 0 résultats: 0
rang: 2 rang: 1 résultats: 2
rang: 2 rang: 2 résultats: 1
rang: 2 rang: 3 résultats: 2
rang: 3 rang: 0 résultats: 6
rang: 3 rang: 1 résultats: 2
rang: 3 rang: 2 résultats: 14
rang: 3 rang: 3 résultats: 2
rang: 4 rang: 0 résultats: 0
rang: 4 rang: 1 résultats: 3
rang: 4 rang: 2 résultats: 1
rang: 4 rang: 3 résultats: 3
rang: 5 rang: 0 résultats: 6
rang: 5 rang: 1 résultats: 3
rang: 5 rang: 2 résultats: 14
rang: 5 rang: 3 résultats: 3
rang: 6 rang: 0 résultats: 0
rang: 6 rang: 1 résultats: 4
rang: 6 rang: 2 résultats: 1
rang: 6 rang: 3 résultats: 4
rang: 7 rang: 0 résultats: 6
rang: 7 rang: 1 résultats: 4
rang: 7 rang: 2 résultats: 14
rang: 7 rang: 3 résultats: 4
rang: 8 rang: 0 résultats: 0
rang: 8 rang: 1 résultats: 5
rang: 8 rang: 2 résultats: 1
rang: 8 rang: 3 résultats: 5
rang: 9 rang: 0 résultats: 6
rang: 9 rang: 1 résultats: 5
rang: 9 rang: 2 résultats: 14
rang: 9 rang: 3 résultats: 5
rang: 10 rang: 0 résultats: 0
rang: 10 rang: 1 résultats: 6
rang: 10 rang: 2 résultats: 14
rang: 10 rang: 3 résultats: 10
```

On remarque que le tableau a été réduit de 14 à 10, cela n'est pas impressionnant car ce test a été réalisé avec seulement la possibilité de concaténer juste 2 lignes du terrain entre elle avec un coordonnée y différent.

Si vous activez la boucle ligne 370 de la classe Terrain :

```
//while (m < 20) {
```

→ La listeRectangleATondre ne comportera plus que les blocs à tondre comme ceci :

```
-5999999999999999
9999999999999999
9933339999999999
9933339999999999
9933339999999999
9933339999999999
9999999999999999
9999999999999999
9999999999999999
9999999999999999
9999999999999999
9999999999999999
```

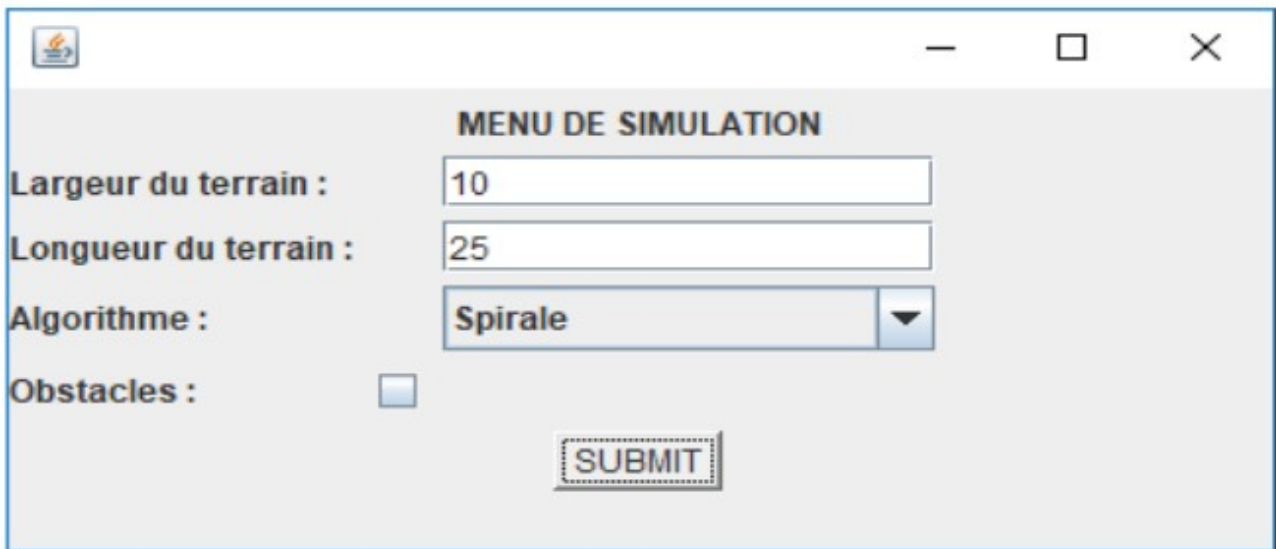
Cette méthode retourne la liste `RectangleATondre`, on peut donc directement appliquer dessus une méthode qui permettra au robot de se rendre au coordonnée  $x = 0$  et  $y = 0$  de chaque rectangle et ainsi d'en effectuer la tonte.

Les possibles améliorations de cet algorithme sont de trouver une solution afin que l'algorithme soit plus rapide pour tenter de grouper les lignes entre elles afin que l'on est des blocs avec plus de 2 lignes rapidement (actuellement cela est long) ainsi que d'améliorer la rapidité en général de cet algorithme en diminuant le nombre de boucle.

## 2. Interface graphique

Tout d'abord en lançant l'application le menu suivant s'affiche :

Comme on peut voir sur le menu, l'utilisateur peut saisir la taille de son terrain, les tailles sont entrées en mètres et sont converties en pixels.



The screenshot shows a window titled "MENU DE SIMULATION". It has four input fields: "Largeur du terrain :" with the value "10", "Longueur du terrain :" with the value "25", "Algorithme :" with a dropdown menu showing "Spirale", and "Obstacles :" with an unchecked checkbox. At the bottom center is a button labeled "SUBMIT".

Ensuite il choisit le type de son algorithme, Normal, Spirale, ou Random, qui est récupéré par le menu afin de lancer le bon algorithme.

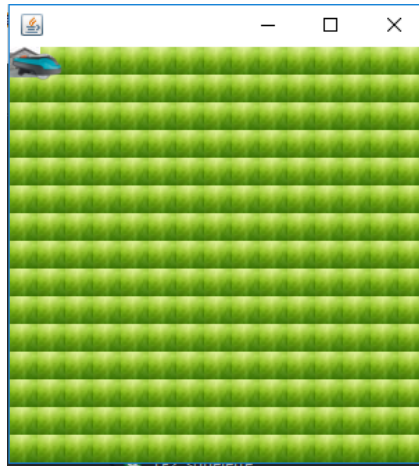
Une fois les données saisies l'utilisateur n'a qu'à appuyer sur le bouton submit et la simulation demandée se lance.

Le robot commence dans la première ligne à l'extrémité gauche et commence à bouger en fonction de l'algorithme choisis .

N.B : Un `thread.sleep ()` est placé après chaque mouvement pour que l'utilisateur puisse bien visualiser le déroulement de la simulation .



On considère que la case départ représente la base de chargement du robot donc on aura pas à l'éviter vu qu'on ne repassera plus dessus à part sur l'algorithme Random.



#### a) Initialisation des images du terrain

```
public void paintComponent(Graphics g) {  
  
    super.paintComponent(g);  
    setBackground(Color.WHITE);  
    g.setColor(Color.BLACK);  
    try {  
        fond1 = ImageIO.read(new File("gazon_tondu.jpg"));  
        fond2 = ImageIO.read(new File("gazon.jpg"));  
        fond3 = ImageIO.read(new File("robot2.png"));  
        fond4 = ImageIO.read(new File("garage-1.jpg"));  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
    g.setColor(Color.GRAY);  
  
    g.drawImage(fond2, x, y, this);  
  
    for (int i = 0; i < lon; i += 20) {  
        for (int j = 0; j < lar; j += 20) {  
            if (t[j / 20][i / 20] == 0) {  
                g.drawImage(fond1, i, j, this);  
            }  
            if (t[j / 20][i / 20] == 1) {  
                g.drawImage(fond2, i, j, this);  
            }  
        }  
    }  
  
    g.drawImage(fond4, 0, 0, this);  
    g.drawImage(fond3, x, y, this);  
  
}
```

Dans cette partie du code on charge les images tout d'abord, ensuite on rentre dans la boucle, si la case a été découverte on affiche l'image du gazon tondu sinon on affiche l'image du gazon non tondu, on répète cela jusqu'à ce qu'on couvre tout le terrain.

En ce qui concerne les deux dernières lignes du code, la première sert à afficher l'image de la base de chargement à la première case, et la seconde permet d'afficher l'image du robot à sa position actuelle pour qu'il ne soit pas caché.

À chaque fois qu'on fait un repaint(), la même procédure se reproduit.

## b) L'algorithme Spirale

```
// Algo spirale
public void algo_spirale() throws InterruptedException {

    //la limite gauche de l'interface
    SpiG = 0;
    //la limite droite de l'interface
    SpiD = lon - 20;
    //la limite du haut de l'interface
    SpiH = 20;
    //la limite du bas de l'interface
    SpiB = lar - 20;

    Thread.sleep(1000);

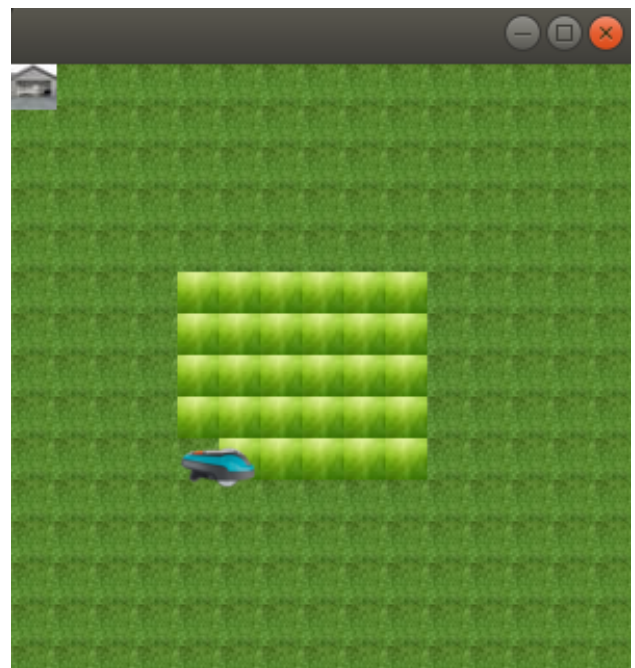
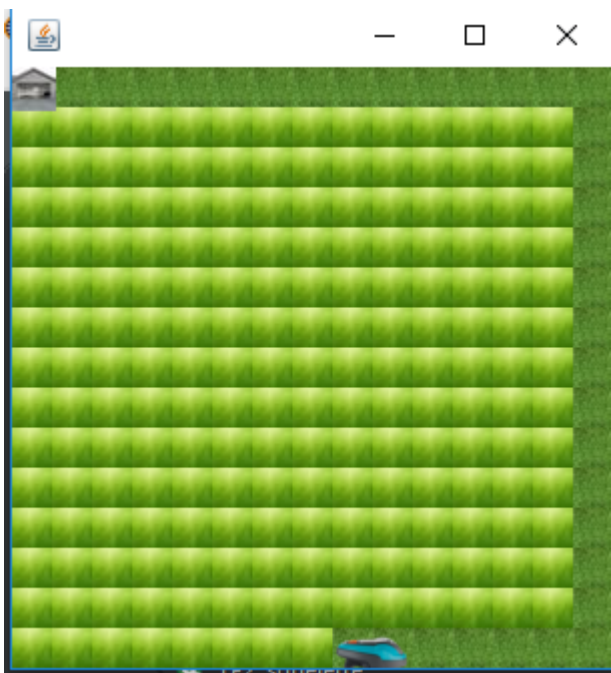
    while (cmtSpi != a * b) {
        while (x < SpiD) {
            Droite();
            Thread.sleep(100);
        }
        while (y < SpiB) {
            Bas();
            Thread.sleep(100);
        }
        while (x != SpiG) {
            Gauche();
            Thread.sleep(100);
        }
        while (y != SpiH) {
            Haut();
            Thread.sleep(100);
        }

        // changement des bornes du terrain 0 tondre (la zone devient plus petite)
        SpiG += 20;
        SpiD -= 20;
        SpiH += 20;
        SpiB -= 20;
    }
}
```

Pour cet algorithme son fonctionnement a déjà été expliqué sur la partie console.

La variable `cmtSpi` représente le nombre de cases sur le terrain, tant que ce nombre n'a pas été atteint on continue à exécuter l'algorithme.

On commence par une boucle qui parcourt tout le ligne de droite à gauche, une fois arrivé à la limite du terrain, on passe à une autre boucle pour parcourir la ligne suivante du haut vers le bas en partant de la case d'arrivée de la première boucle, pour la troisième boucle on parcourt la ligne qui commence de la case d'arrivée de la précédente boucle en allant de gauche à droite, arrivant à la dernière case qui est la limite du terrain au coin bas gauche, on passe à la dernière boucle qui elle parcourt la ligne qui commence de la case au coin gauche en allant vers le haut mais qui s'arrête dans la case se situant juste avant la ligne qu'on a parcouru au tout début, une fois fait, change les bornes du terrain pour tondre une nouvelle zone et ainsi de suite jusqu'à ce qu'on tond tout le terrain.



### c) L'algorithme Normal

Le placement est le même que celui d'avant. Cet algorithme est assez simple comme vous avez pu voir lors de son explication avant. Le but c'est de parcourir la ligne et une fois arrivé à la limite du terrain on passe à la ligne d'après en passant à la cellule se situant en dessous de la cellule d'arrivée.

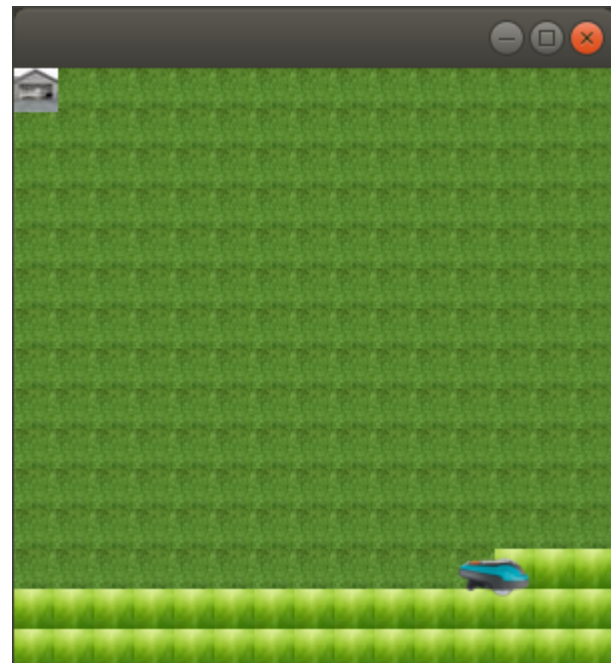
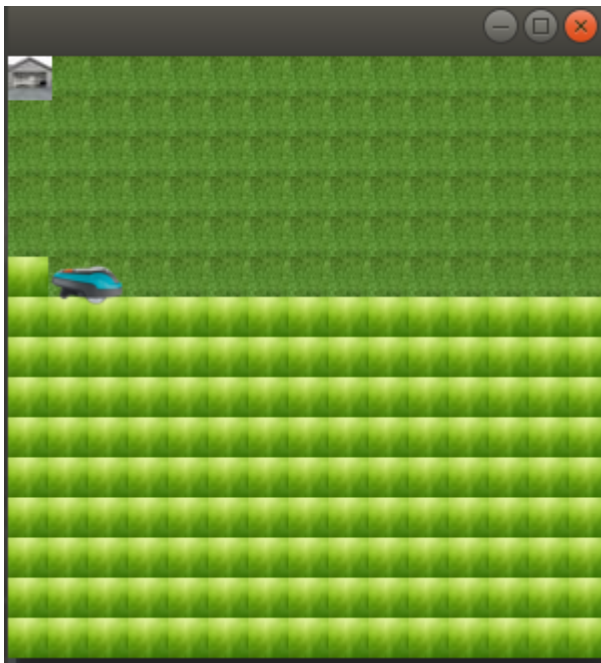
```
// algo normal
public void algo_normal() throws InterruptedException {

    Thread.sleep(600);
    while (cmt != b) {

        while (x <= lon - 40) {
            Droite();
            Thread.sleep(100);
        }
        Bas();
        while (x != 0) {
            Thread.sleep(100);
            Gauche();
        }

        Thread.sleep(100);
        Bas();
        Thread.sleep(100);
        cmt++;
    }
}
```

En lançant l'algorithme on obtient le résultat suivant :



#### d) Algorithme Random

Pour celui ci on a rencontré des problèmes de conception car on ne peut pas tirer des angles au hasard alors que notre interface se base sur des pixels, ce qui fait que cet algorithme est toujours en phase de conception et de tests avant de pouvoir aboutir à une solution raisonnable et efficace.

## IV Bilan

Et si c'était à refaire ? Coût + techniques ?

En parlant des coûts, le projet n'a absolument rien coûté, tout le travail a été fait sur nos propres machines et sur des logiciels et IDE gratuits à 100 %. La seule chose qui aurait coûté de l'argent c'est les cartes Arduino ainsi que le robot pour implémenter nos fonctions dessus et lancer une vraie simulation.

Du côté des techniques, on aurait utilisé quelques outils de développements gratuits ( Windows Builder, Turtle ...) disponibles sur Eclipse (ou à télécharger), qui pourraient accélérer notre travail en groupe ou seul.

On aurait pensé à utiliser le Turtle sur Python pour pouvoir faire les différents types d'algorithmes possibles mais pour cela il faut encore plus de temps à fin de maîtriser le langage à nouveau et l'outil aussi vu que c'est la première fois qu'on l'utilise.

On aurait une idée précise sur la façon de conception et donc on optimisera un maximum notre travail, car c'est vrai que tout nous semblait flou au début, on aurait eu une nouvelle stratégie de développement plus précise et ciblée car tout est clair devant nous maintenant.

On aurait pris le temps de chercher des astuces dans des livres de programmation ou dans des blogs sur internet plus que ce que l'on a fait vu qu'on était pressé par le temps cette fois-ci, entre la compréhension du but du projet la manière de développement ce qui est attendu et tout ça on aurait beaucoup moins de mal à nous situer dans le projet qu'avant.

On aurait compléter les fonctionnalités ainsi qu'on aurait enrichi un maximum notre interface graphique par des obstacles et des algorithmes différents ce qui était le point noir pour nous sur le projet car on avait du mal à résoudre les bugs et le manque de tutoriels sur ce sujet sur internet.

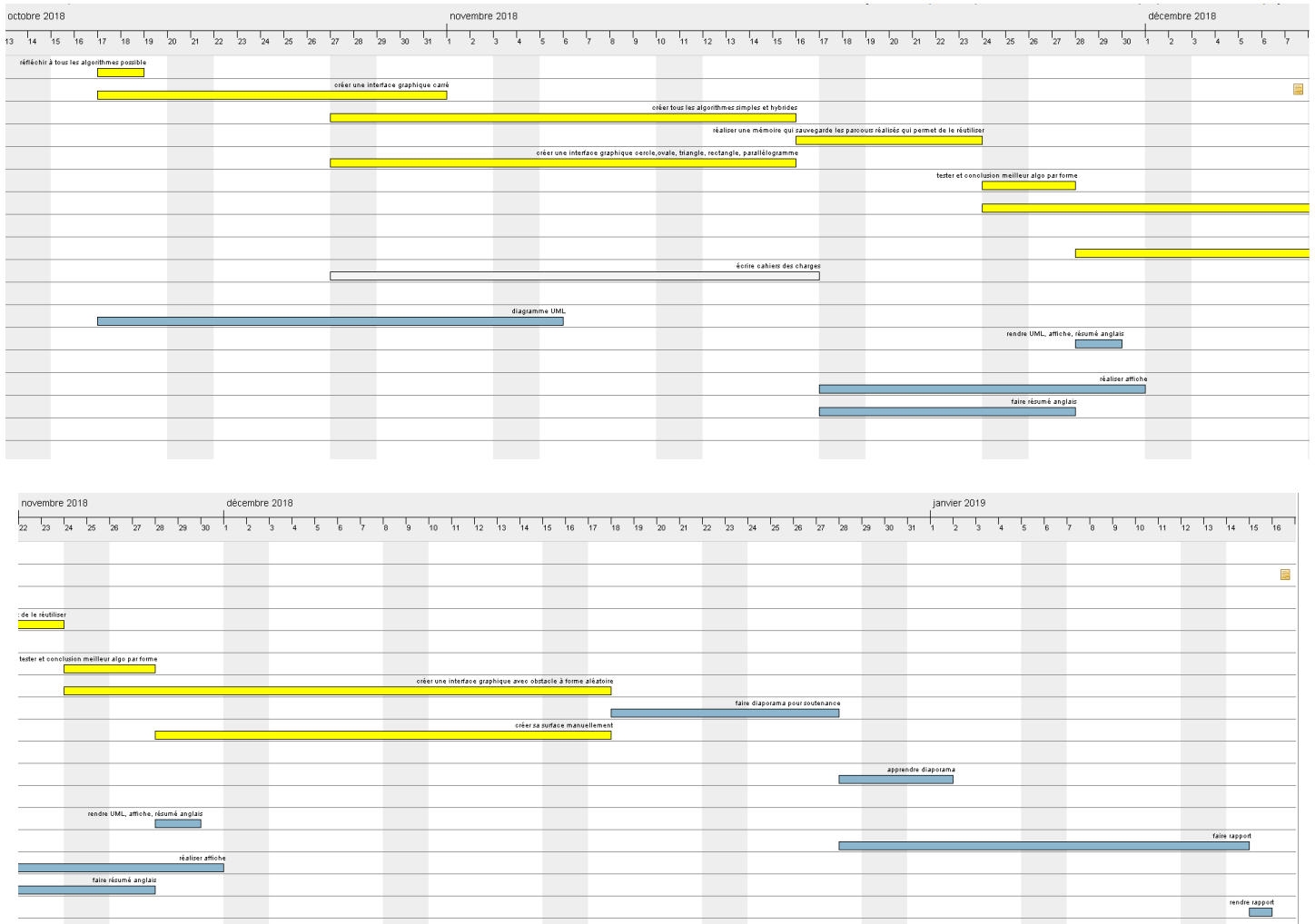
## IV. Perspectives

Ce projet nous laisse d'énormes perspectives pour sa poursuite, que ça soit niveau développement ou recherche ou autre ..

- Tout d'abord, avec l'ajout de la fonction de création de rectangles sur le terrain, on aurait compléter le code en ajoutant le code de l'algorithme A\* pour trouver le plus court chemin entre les rectangles et le passer au robot pour aller plus vite et tondre plus efficacement lors de l'existence des obstacles.
- On avait aussi voulu rajouter un mode de simulation utilisant plusieurs robots à la fois, avec l'utilisation des Thread on aurait pu mettre en route deux robots ou plus sur le même terrain, mais vu qu'on a pas eu de cours sur cette partie là c'était un dur à comprendre pour nous, mais ce la reste une perspective car après tout il nous a fallu juste du temps pour réussir à l'implémenter.
- On aurait aussi compléter notre interface graphique et la rendre plus cohérente et plus attirante pour l'utilisateur, et pourquoi pas respecter les règles d'ergonomie. On pourrai également développer le programme pour pouvoir lancer plusieurs simulation à la fois à fin de faire des comparaisons en temps réel entre les différents algorithmes disponibles.
- Ensuite, on pourrai rajouter des fonctionnalités à notre main, par exemple le contrôle de la vitesse du déroulement, l'ajout des obstacles manuellement, une fenêtre apaisante pour l'utilisateur et facilite la prise de décision pour lui vu que toutes les informations seront bien ranger et organisées.
- Par ailleurs, sur l'interface graphique cette fois-ci il y a la possibilité de rajouter un panel qui affiche des informations tel que le temps mis, la vitesse du robot, la surface de la zone tondue, l'efficacité en pourcentage de l'algorithme choisis pour ce type de terrain, une option de zoom avant ou arrière au cas où le terrain est super grand, possibilité de changement de vitesse pendant le déroulement de l'algorithme si l'utilisateur ne veut pas attendre beaucoup de temps.
- Il existe d'autres informations à rajouter sur le panel de l'interface tel que le pourcentage de batterie et sa consommation pendant la tonte du terrain, possibilité d'arrêter le robot dans un endroit du terrain à l'aide d'un bouton stop, et enfin rajouter des obstacles encore une fois dans la place désirées et non pas automatiquement par le programme.
- Il nous reste la possibilité de réaliser des tests unitaires sur les algorithmes du mode console pour éviter tout les bugs possible et gérer toute situation possible dans un terrain, par exemple la boue, les trous, une grande flaque d'eau, un petit bassin, des montée..
- On peut aussi améliorer le code et le traduire en Arduino pour pouvoir le mettre sur une carte et tester sur un vrai robot pour voir si c'est vraiment rentable ou pas pour chaque algorithmes et veiller sur le bon fonctionnement de ces derniers.

# V. ANNEXES

## 1. Planning

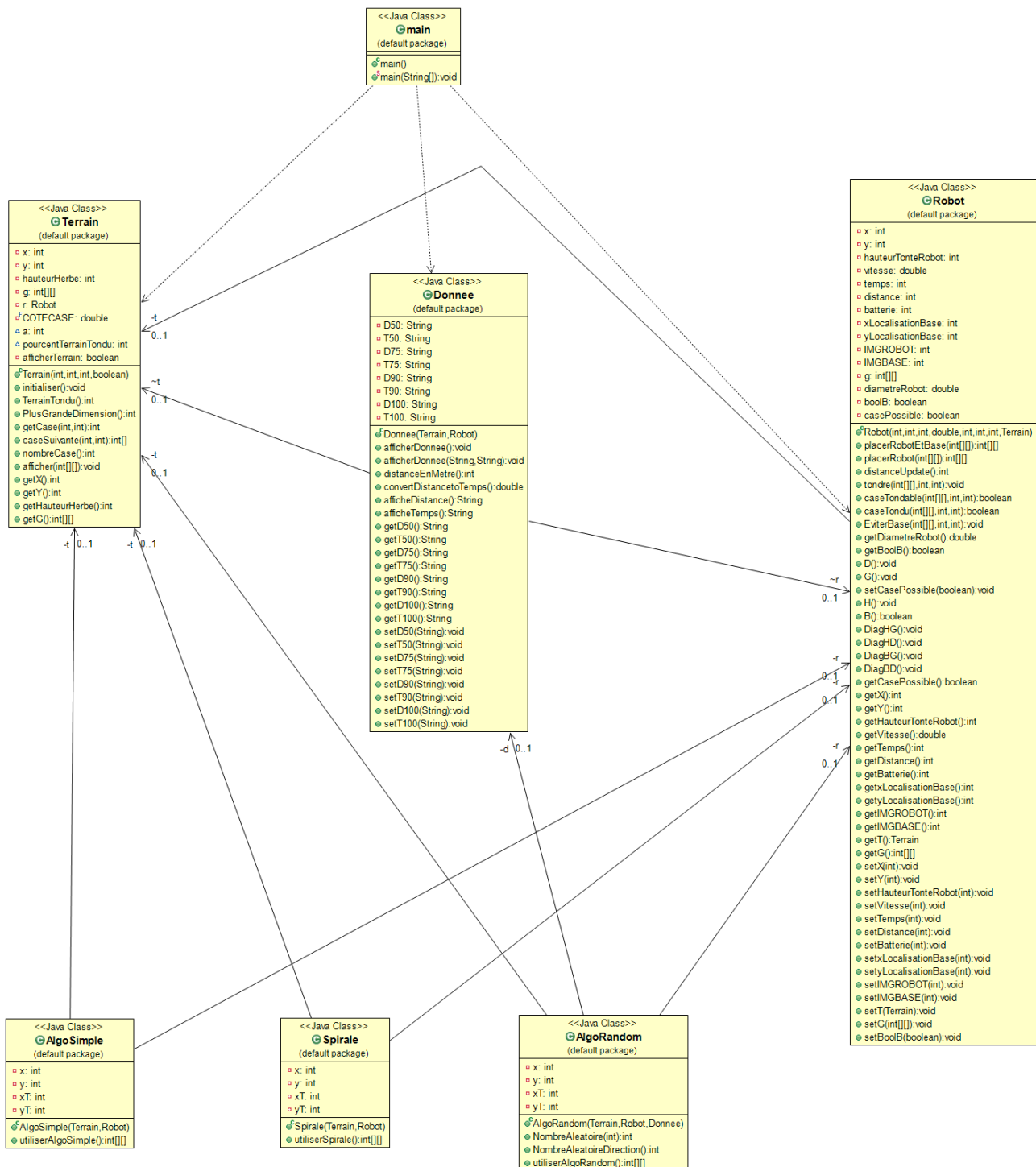


Comme vous avez pu le voir durant tout ce rapport, le programme initial n'a pas été complètement respecté, cela est dû à des complications de mise en place des algorithmes ainsi que de la partie graphique. De plus, nous n'avons que peu de temps libre pour se consacrer pleinement à ce projet ce qui rend la tâche plus pénible pour nous et nous rajoute de la pression.



## 2. Diagrammes UML

Ceci est notre diagramme UML concernant la partie Console, il représente les différentes classes utilisées ainsi que leurs composants.





### 3. Notice du main

#### a) Partie console

Premièrement tout se passe dans le fichier main.java

Le voici :

```
public static void main(String[] args) {

    1.// Terrain(int x, int y, int hauteurherbe, boolean affichage)
    2.Terrain t = new Terrain(5,5,9, true)
    3.//public Robot(int x, int y, int hauteurTonteRobot, double vitesse, int
    batterie, int xLocalisationBase,int yLocalisationBase, Terrain t)
    4.Robot r = new Robot(0, 0, 1, 0.4 ,100, 9, 4, t);
    //public Donnee(Terrain t, Robot r)
    5.Donnee d = new Donnee(t, r);
    // AlgoSimple(Terrain t, Robot r)
    6.AlgoSimple aS = new AlgoSimple(t, r);
    // Spirale(Terrain t, Robot r)
    7.Spirale sp = new Spirale(t, r);
    // AlgoRandom(Terrain t, Robot r, Donnee d)
    8.AlgoRandom aR = new AlgoRandom(t, r, d);
    // initialiser le terrain
    9.t.initialiser();
    // afficher le terrain
    10.t.afficher(t.getG());
    //placerObstacle(double x1, double y1, double x3, double y3)
    /*x1 et y1 sont les coordonnées du sommet en haut à gauche, x3 et y3 en
bas à droite*/
    11.t.placerObstacle(1,1,3,3);
    //r.placerRobotEtBase(t.getG());
    12.r.placerRobot(t.getG());
    13.t.afficher(t.getG());
    14.//aS.utiliserAlgoSimple();
    15.//sp.utiliserSpirale();
    16.aR.utiliserAlgoRandom();
}
```

Les numéros sur le côté gauche sont les numéros de ligne du code.

Les caractères // ou /\* \*/ représentent l'explication de la fonction en dessous sauf pour les lignes 14,15 où les symboles // signifient que la fonction est désactivé.

On ne peut que utiliser un algorithme à la fois. Pour désactiver un algorithme il suffit de placer devant lui : //

Pour activer un algorithme il suffit de retirer // placé devant lui.

Pour initialiser une classe par exemples le Robot il suffit de rentrer les données nécessaires suivi d'une virgule comme ceci :

```
Robot r = new Robot(0, 0, 1, 0.4 ,100, 9, 4, t);
```

Les données qui sont placés correspondent au commentaire au dessus :

```
//public Robot(int x, int y, int hauteurTonteRobot, double vitesse, int  
batterie, int xLocalisationBase,int yLocalisationBase, Terrain t)
```

La hauteur de la tonte du Robot est la hauteur avec laquelle il va placer sa lame, la batterie signifie le pourcentage d'énergie qu'il y a dans la batterie.

Si le terrain est grand il vaut mieux que la simulation s'arrête à 50 % ou 70 %, pour cela il suffit de se rendre dans la classe AlgoRandom, tout en bas de l'algorithme il faut activer le break dans le cas du 50 % de tondu si vous voulez que la simulation s'arrête à 50 % de tondu.

```
if(t.TerrainTendu()== 50 ) {  
  
    d.setD50(d.afficheDistance());  
    d.setT50(d.afficheTemps());  
    break;  
  
}
```

## b) Partie graphique

pour le main de la partie graphique il en existe deux soit avec une interface soit sur une console, comme le main en interface a déjà été expliqué il nous reste donc à voir la partie du main en console.

Tout d'abord, on vous présente le code du main :

```
package Simple;

import java.awt.EventQueue;

public class Plateau2Main {
    public static void main(String[] args) throws InterruptedException {

        int longueur, largeur;
        Scanner input = new Scanner(System.in);

        System.out.println("Entrez la longueur de votre terrain en mètres : ");
        longueur = input.nextInt() * 60;

        System.out.println("Entrez la largeur de votre terrain en mètres : ");
        largeur = input.nextInt() * 60;

        Plateau2 f = new Plateau2("ROBOT TONDEUSE SIMULATION", longueur, largeur);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.pack();
        f.setVisible(true);
        f.algo_spirale();
        //f.algo_normal();
        //f.algo_random();
    }
}
```

Tout au début on commence par demander les proportions du terrain à l'utilisateur, toujours en mètres, ces valeurs sont multipliées par 60 pour avoir le bon affichage sur l'interface, et sont passées par la suite au constructeur du Plateau, qui prend en paramètre:

- Un string : le titre de la fenêtre
- Longueur et largeur du terrain ( après multiplication par 60).

Ensuite on a les fonction reliées à la classe pour pouvoir bien afficher la fenêtre.

Et juste en dessous on retrouve les différentes fonction de l'interfaces ( algorithmes possibles).

On peut faire plusieurs création et plusieurs simulation en même temps si on créé d'autres plateaux et que l'on les appelle avec l'une de ces fonctions, dans ce cas c'était l'algorithme Spirale.

Les résultats sont les mêmes que ce qu'on a vu avant, il n' ya que la manière de lancement du programme qui change.

## **4. Documentation JAVADOC**

La documentation étant trop grande avec de grands tableaux, les captures d'écran n'auraient pas bien été présentées.

La documentation de tous les fichiers est disponible dans le fichier joint Documentation.tar.gz.

## **5. Cahier des charges**

### **1. Contexte et définition**

Ce projet existe dans le cadre où depuis quelque temps, un nouveau moyen de couper son herbe existe : le robot tondeuse. Avant la conception du robot en lui-même, il faut développer sa partie simulation informatique. Le robot est chargé de couper l'herbe dans toute surface tout en économisant le temps et l'énergie. Notre projet consiste à développer des algorithmes permettant d'optimiser le temps consommé ainsi que la qualité de tonte.

### **2. Objectif du projet**

L'objectif est de développer un programme de simulation capable de tester en temps réel n'importe quel terrain avec un robot en utilisant des caractéristiques définies. Le but étant de développer les méthodes de déplacement les plus rapides ainsi que pouvoir évaluer le temps pour couvrir le terrain en entier. Ce programme devra permettre d'avoir une tonte optimisée pour chaque type de terrain.

### **3. Périmètre du projet**

Le projet comportera uniquement une simulation informatique d'une interface qui représente le terrain, d'un robot, des obstacles sur le terrain, et de paramètres à calculer par rapport à ce dernier. Il n'y aura pas de développement du logiciel pour implémenter sur le robot.

### **4. Description fonctionnelle des besoins :**

Le projet doit répondre aux besoins suivants :

- Lancer la simulation du robot et du terrain, avec possibilité de placer des obstacles de différentes formes ( triangle, cercle, carré...)
- Calculer le temps mis pour tondre toute la surface, enregistrer chaque déplacement et pouvoir analyser lequel était le plus court
- Adapter un algorithme en fonction du type et la forme du terrain
- Système de gestion des zones déjà tondues pour optimiser le temps consommé
- Pouvoir fournir le schéma du terrain par le client, ce qui permettra de lui fournir une simulation de tonte pour son terrain.

- Améliorer la forme des obstacles pour l'adapter à la vraie vie.

## **5. Enveloppe budgétaire**

Le projet est réalisé bénévolement par DELANDHUY Mattéo & AISSI Ayoub, dans le cadre du projet tutoré 3 du 3ème semestre en DUT Informatique. Il n'y aura pas de matériel à utiliser, tout les programmes seront réalisés sur l'IDE Eclipse.

## **6. Délais de réalisation**

L'application devra être livrée au plus tard pour le 15 janvier, cela permettra d'implémenter les changements suggérés par le jury lors de la soutenance du projet pour livrer un programme optimisé.

## **6. Codes sources**

Vu que les codes sont trop grands et volumineux, ils sont présents dans l'archive CodesSources.tar.gz

## **7. Rapport en anglais**

Voir le fichier RapportAnglais.odt