Rapport du projet-Info
Simulation de déplacement programmé et aléatoire dans un espace théorique

Table des Matières :

Table des Matières : 2
Table des Figures : 2
Introduction : 3
1.1- Règles et principes : 3
1.2- Pour R : 3
1.3- Pour T : 3
2- Voisinage de Moore d'ordre 1 : 3
3.1- Début du code, déclaration des variables et obtention des informations : 4
4.1- Boucle générale et vue globale des tondeuses R et T : 5
4.2- Quels avantages et inconvénient ? 6
5- Fonction « ValeurAleatoire » : 7
6.1- Etude de l'exécution du code Global : 9
6.2- Mais pourrait-il y avoir un changement si on déplace R dans d'autres coins ? 9
6.3- Mais pourrait-il y avoir un changement si on déplace T ? 10
7- Etude de R : 11
8.1- Etude de T : 12
8.2- Comment évolue le temps T avec la tondeuse T ? 13
9.1- Etude de la grille et de son implication dans les performances : 13
9.2- Etude de la grille avec R : 13
9.3- Etude de la grille avec T : 14
9.4- Etude de la grille avec R et T : 15

Table des Figures :

Figure 1 : Histogramme des scores 9
Figure 2 : Histogramme de comparaison des scores de simulation avec différents placements de R .. 9
Figure 3 : Histogramme de comparaison des scores avec différents placements de T 10
Figure 4 : Histogramme de comparaisons des scores en temps T des simulations aléatoire 13
Figure 5 : Histogramme des résultat de R avec une grille personnalisable 13
Figure 6: Histogramme des moyennes arrondie de T avec une grille personnalisable sur 2 tirages ... 14
Figure 7 : Histogramme des moyennes de T, R , et de Temps avec un terrain de plus en plus grand représenté par l'Air. 15
Figure 8 : Histogramme des moyennes de Temps normal et de Temps hypothèse 16

Introduction :

Le premier objectif de ce projet était de concevoir un espace théorique en 2 dimensions représentant un terrain, et remplie à l'initialisation d'une ressource " ~ " considéré comme du gazon.
Le second objectif était donc de tondre ce gazon à l'aide de deux robot tondeuse, T et R. Chaque tondeuse à son propre fonctionnement mais elles doivent être lancées en même temps sur le même tableau à deux endroits différents.
La tonte du terrain sera définie par le remplacement de " ~ " par " _ ", augmentant le score et chaque mouvement se fera en temps T+1.
Et enfin R à une trajectoire prédéfinit ligne par ligne, tandis que celle de T est aléatoire et définit selon un voisinage de Moore d'ordre 1.

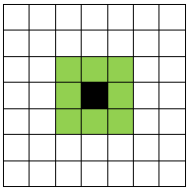
1.1- Règles et principes :

Pour commencer le projet, la première chose à faire fut d'établir des règles pour chaque tondeuse afin de respecter les consignes du projet :

1.2- Pour R :
-R ne peut tondre T et devra attendre son départ ;
-R ne commencer que dans l'un des 4 coins de la grille (au choix) ;
-R ne s'arrête pas pour une parcelle déjà tondue et ne gagne pas de point pour cette dernière ;

1.3- Pour T :
-T ne peut tondre R et devra attendre son départ ou d'avoir une autre direction ;
-T, du fait de ses mouvements aléatoires, as ses déplacements limités par les bords du terrain ;
-T pourra à la fin commencer partout sauf sur R ;
-T ne s'arrête pas pour une parcelle déjà tondue et ne gagne pas de point pour cette dernière ;

2- Voisinage de Moore d'ordre 1 :



Un voisinage de Moore est une règle de déplacement concernent les points des grilles 2 dimensions, ces derniers ne pourrons se déplacer que dans les 8 case adjacente au point. Nous avons donc un total de 8 directions à faire choisir aléatoirement dans ce projet. Par conséquent j'ai décidé d'utiliser le Mersenne Twister avec un intervalle [0,1].

Dans cette présentation le code ne sera pas retranscrit à l'état brut mais simplifié, et ce principalement dû au fait qu'il soit d'une très grande taille, il sera cependant donné en annexe. Entièrement commenté.

De plus pour avoir un tirage aléatoire différent d'un lancement de programme à l'autre la clé d'initialisation du Mersenne Twister doit être changer depuis le code.

3.1- Début du code, déclaration des variables et obtention des informations :

Dans un premier temps nous devons mettre en place les ressources du code. Ici nous introduisons les librairies ainsi que l'implémentation du code externe "MT.c" qui contient le programme Mersenne Twister.
La suite vient directement du début du main, ou on se concentre principalement sur la création et ou la récupération de variables essentiel, ainsi que sur l'interface utilisateur :

Globalement le code suit cette simplification :

```
Déclaration des librairies ;

Intégration de MT.c ;

main(vide)
{
        Initialisation du Mersenne twister via l'envoie de clé ;

        Déclaration et initialisation des variables nécessaires ;
        Déclaration des variables score pour R et T à 1 ;
        Déclaration des variables mode pour R et T à -1 ;

        Affichage de l'interface et des propositions de paramètre ;
        Récupérations des données entrées via des boucles de vérification ;

        Déclaration du T de temps à 0 ;
        Affichage du tableau en T = 0 sans tondeuse ;


        Placement de la tondeuse R en fonction des choix des paramètres de modeR
        {
                Initialisation de la variable GouD de direction ;
                Placement ou non de R ;
        }

        Condition en fonction des choix des paramètres de modeT
        {
                Initialisation des variables de position de T
                Placement ou non de T ;
        {

        Affichage du tableau en T = 0 avec tondeuse ;
        T de temps prend +1 ;
        Déclaration et initialisation des variables nécessaire ;
        Attribution de NBaleatoire = le retour de la fonction ValeurAleatoire ;
```

On utilise ici de nombreuses boucle « while » pour le bon fonctionnement du code mais aussi afin de limiter l'utilisateur lors de son intervention. On veut ici que l'utilisateur donne uniquement des valeurs valide aux yeux du code.

4.1- Boucle générale et vue globale des tondeuses R et T :

Le code continue avec la programmation des IA des tondeuses R et T. Nous étudierons plus tard et plus en détail, chaque programme de tondeuse.

Voici la simplification globale du programme :

```
Boucle globale du code tant que (score R + score T) < score final possible
{
        Affichage de T en temps ;
                Double boucle de déplacement dans le tableau pour R
                { //début boucle 1 { //début boucle 2
                        Nombreuses conditions pour avoir le bon déplacement de R
                        /*En fonction de modeR principalement*/
                        {
                                Déplacement ou non de R ;
                                Ajout ou non de score en fonction du type d'herbe ;
                                Attribution d'une valeur à dejaFait ;
                                Blocage si T déjà présent ;
                                On sort de la boucle ;
                        }
                } //fin boucle 2
                        Condition pour dejaFait afin d'éviter répétition
                        {
                                dejaFait est remis à 0 ;
                                On sort de la boucle ;
                        }
                } //fin boucle 1
                Double boucle de déplacement dans le tableau pour T
                { //début boucle 1 { //début boucle 2
                        Nombreuses conditions pour avoir le bon déplacement de T
                        /*En fonction de la direction NBaleatoire principalement*/
                        {
                                Déplacement ou non de T ;
                                Ajout ou non de score en fonction du type d'herbe ;
                                Attribution d'une valeur à dejaFait ;
                                Blocage si R déjà présent ;
                                On sort de la boucle ;
                        }
                } //fin boucle 2
                        Condition pour dejaFait afin d'éviter répétition
                        {
                                dejaFait est remis à 0 ;
                                On sort de la boucle ;
                        }
                } //fin boucle 1
        T de temps prend +1 ;
        Double boucle d'affichage du tableau ;
}
Affichage du score final ;
} FIN du main
```

Ici le code suit une logique d'imposition de multiple condition afin d'avoir la capacité d'agir lors de chaque situation possible.
Chaque tondeuse agis dans la même boucles « while », mais elles possèdent leurs propres doubles boucles « for », c'est un choix volontairement prit, cette optique à un léger impacte sur la rapidité du code mais permet en échange de faciliter la visibilité de ce dernier et limiter les erreurs et les variables inutiles.

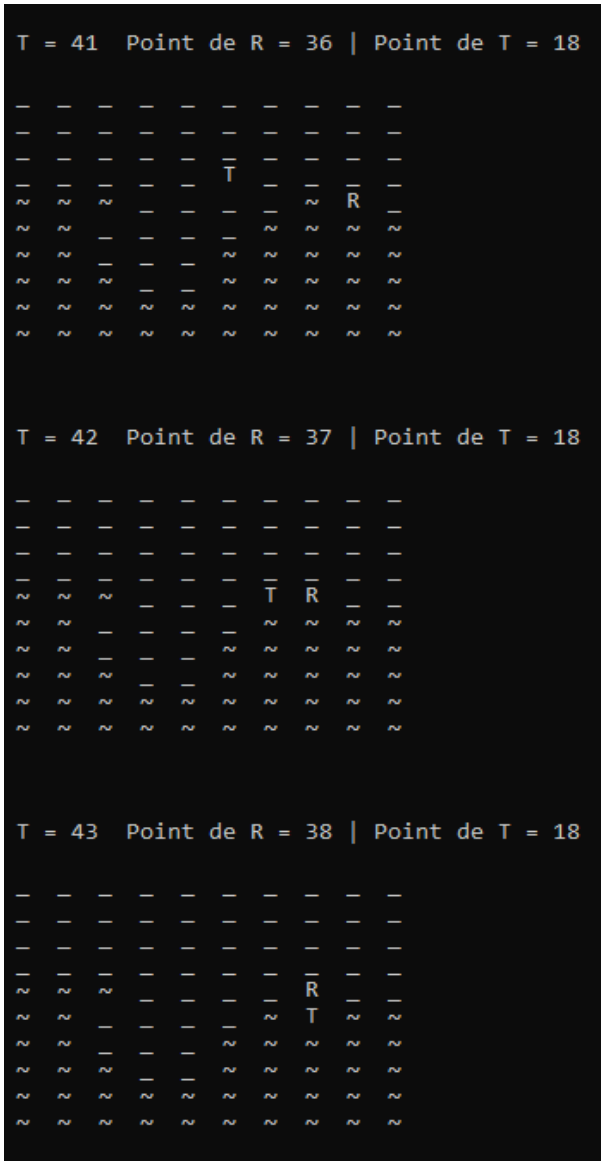


Image 1 : Blocage de R par T

4.2- Quels avantages et inconvénient ?

Cependant que les tondeuses soient dans la même boucle while et for ou non, il y aura évidemment toujours une tondeuse qui sera lancé en première et qui aura donc l'avantage ou l'inconvénient du premier tour, ici ce sera R qui tournera en premier.

Ça peut être un inconvénient pour R :

Cela peut signifier que si T se trouve sur le chemin de R, R ne bougera pas, même si T se déplace juste après le tour de R.

Ça peut être un avantage pour R :

Cela peut aussi signifier que si T et R on le même objectif, R l'aura en premier et T devra attendre et donc probablement aller ailleurs.

Notons enfin que pour convenir à l'affichage présenté sur le sujet du TP, R et T obtienne 1 point dans leurs scores de tonte après avoir été posé sur une herbe mais ne l'aurons d'afficher sur l'interface qu'après avoir commencé leur nouveau tour.

5- Fonction « ValeurAleatoire » :

La dernière partie du code est constitué de la fonction valeur aléatoire, cette fonction appelle le Mersenne twister et renvoie une valeur en int compris entre 0 et 7 :

```
ValeurAleatoire
{
        Création du int ConvertionNB à 10 pour entrer dans la première boucle ;
        Création du double ReceptionNB ;

        Boucle while fonctionnant tant que ConversionNB >=8
        {
                Attribution de ReceptionNB au retour de la fonction gerand_reall du MT.c ;
                /*gerand_real retourne un nombre à virgule entre 0 et 1*/

                On multiplie ReceptionNB par 10 afin d'avoir un chiffre avant la virgule ;
                ConversionNB prend la valeur de ReceptionNB ;
                /*ConversionNB étant un « int » seul le chiffre avant la virgule compte et sera
                pris*/

                Condition nécessitant ConversionNB > 7
                {
                /*Si le chiffre donné précédemment ne convient pas car > 7 alors on le supprime
                et prend le prochain après la virgule*/

                        Boucle utilisé tant que ConversionNB > 7
                        {
                                On applique la soustraction de ReceptionNB par ConversionNB ;
                                On remultiplie ReceptionNB par 10 ;
                                ConversionNB reprend alors la nouvelle valeur de ReceptionNB ;
                        }

                }
        }
        On retourne ConvertionNB ;
} Fin de ValeurAleatoire
```

Ce code se base sur le fait que le Mersenne Twister renvoie un nombre entre 0 et 1 avec de nombreux chiffre après la virgule, ce code tente donc d'épuiser toutes les ressources de ce nombre au lieu d'en demander un autre, car même si le premier chiffre après la virgule ne convient pas, peut-être que les suivant, données tout aussi aléatoirement, serons eux répondre à ses attentes.

Avec le code source et avec ce rapport se trouvera aussi le programme de simulation nommé « TestNBaleatoire.c » utile pour tester l'efficacité du code « ValeurAleatoire » et observer le nombre de fois où chaque chiffre de 0 à 7 est sortie de ce programme :

Tableau 1 : Etude de la fonction de tirage aléatoire

Tableau 2 : Etude de la fonction de tirage aléatoire

Tableau 3 : Etude de la fonction de tirage aléatoire

On remarque que les résultats sont drastiquement différents d'un essaie à l'autre, le programme envoie donc bien une valeur aléatoire.

6.1- Etude de l'exécution du code Global :

Ici on s'intéresse aux résultats de l'exécution du code, avec les deux tondeuses en même temps.

Commençons par appliquer le mode basique, soit modeR = 1 (début en haut à gauche) et ModeT = 2 (début au centre), avec un graphique de 10*10, on obtient alors sur 10 lancements :

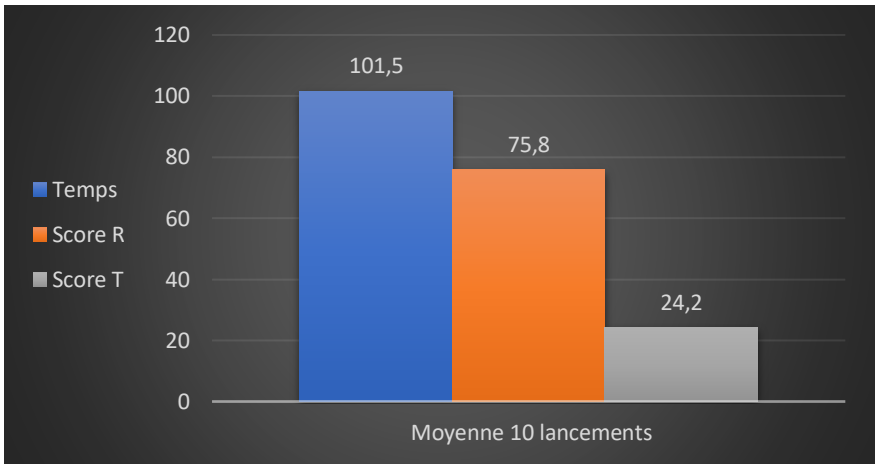


Figure 1 : Histogramme des scores

Nous pouvons observer tout d'abord que le Temp reste proche du nombre d'élément du tableau = 100, ainsi malgré la présence de deux tondeuses différentes dont une au trajet imprévisible, le programme reste suffisamment efficace et n'est ralenti en moyenne que de 1.5 point supplémentaire comparé à R seul.
Cette augmentation du temps peut s'expliquer par la présence de collision entre les deux tondeuses.

Mais il serait donc toujours plus efficace d'équiper deux tondeuse R pour diviser le temp T par deux ou seulement d'avoir R seul.

6.2- Mais pourrait-il y avoir un changement si on déplace R dans d'autres coins ?

La réponse est globalement non dans la majorité des cas :

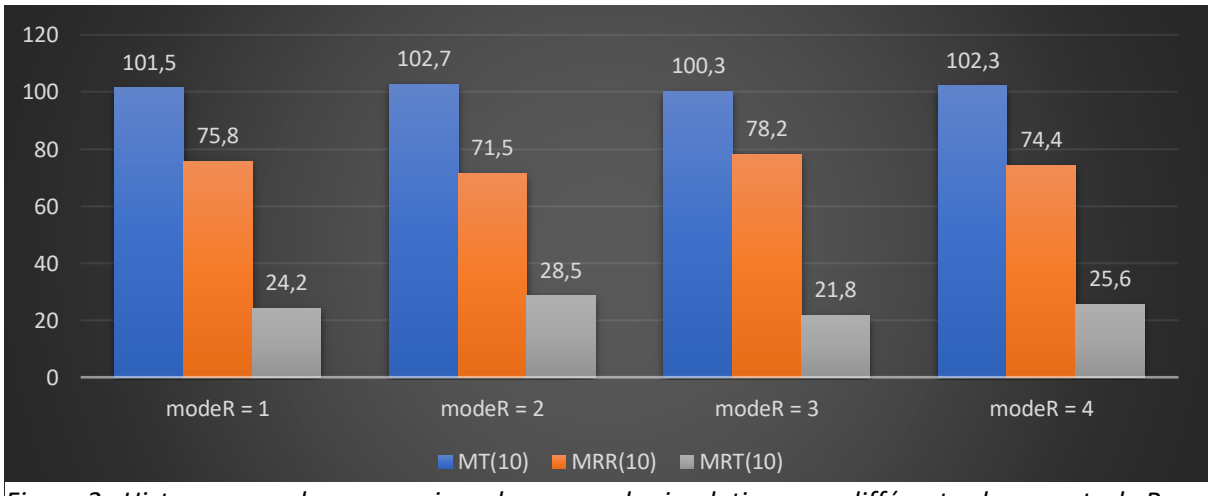


Figure 2 : Histogramme de comparaison des scores de simulation avec différents placements de R

MT(10) = Moyenne de temps sur 10 lancement ;
MRR(10) = Moyenne du Robot R sur 10 lancement ;
MRT(10) = Moyenne du Robot T sur 10 lancement ;

Nous pouvons observer ici que malgré le déplacement de R avec « modeR » le score de temp reste supérieur à 100 et semble très peu varier d'un mode à l'autre.

6.3- Mais pourrait-il y avoir un changement si on déplace T ?

Le principal changement notable s'effectue lorsque l'on approche T de R :

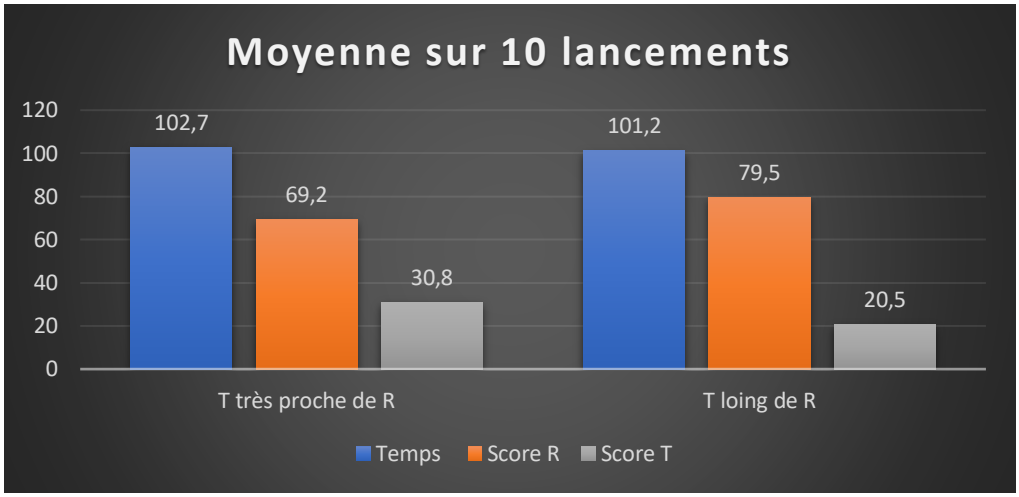


Figure 3 : Histogramme de comparaison des scores avec différents placements de T

Comme on peut le constater R perd de son efficacité et celle de T augmente et inversement lorsque T est loin de R. Cet évènement s'explique par le blocage provoqué par T sur R, R est bloqué à plusieurs instants et ne peut avancer, laissant un avantage à T.

On pourra aussi noter comme perturbation dans l'efficacité de T les placements à côté des limites du tableau, la tondeuse T, en y allant perd au minimum 3 directions aléatoire possible et risque de rester immobile et passer son tour tant qu'une autre valeur valide pour le déplacement ne sera pas récupérée.
Le pire étant que T soit placé dans l'un des 4 coins du tableau, il perdra alors 5 directions possible sur 8.

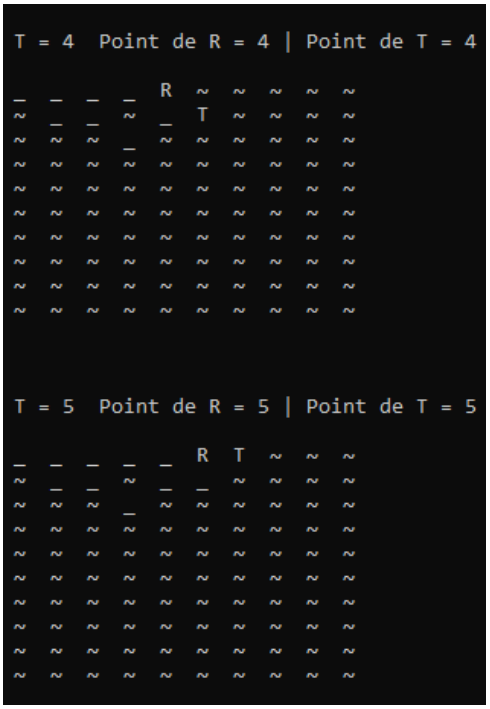


Image 2 : Placement ne laissant pas beaucoup de liberté à T

7- Etude de R :

```
Double boucle de déplacement dans le tableau pour R
        { //début boucle 1
                { //début boucle 2

                        Mise en place des 4 conditions pour le changement de ligne
                        /*Dépend de modeR et du placement de T principalement*/
                        {
                                Déplacement ou non de R ;
                                Ajout ou non de score en fonction du type d'herbe ;
                                Attribution d'une valeur à dejaFait ;
                                Blocage si T déjà présent ;
                                On sort de la boucle ;
                        }
                        Mise en place des 4 conditions pour le déplacement en ligne
                        /*Dépend de modeR et du placement de T principalement*/
                        {
                                Déplacement ou non de R ;
                                Ajout ou non de score en fonction du type d'herbe ;
                                Attribution d'une valeur à dejaFait ;
                                Blocage si T déjà présent ;
                                On sort de la boucle ;
                        }

                } //fin boucle 2

                        Condition pour dejaFait afin d'éviter répétition
                        {
                                dejaFait est remis à 0 ;
                                On sort de la boucle ;
                        }
        } //fin boucle 1
```

Les conditions « if » de déplacement du code de R tourne principalement autour de la valeur de son mode « modeR » et de la position de T. On retrouve aussi des conditions avec pair et impair pour le défilement de haut en bas uniquement.

modeR peut prendre 5 valeurs donnant accès à 5 configurations de R prédéfinit :
0 = Désactivation de R ;
1 = Mise en place de R en haut à gauche de la grille ;
2 = Mise en place de R en haut à droite de la grille ;
3 = Mise en place de R en bas à gauche de la grille ;
4 = Mise en place de R en bas à droite de la grille ;

GouD est aussi une variable très importante et dispose de deux configurations :
1 = R va à gauche
2 = R va à droite
Cette variable est utilisable que pour le défilement de bas en haut

Par sa nature R aura toujours comme score quand il est seul L*C, en un temps L*C.

8.1- Etude de T :

```
Double boucle de déplacement dans le tableau pour T
        { //début boucle 1
                { //début boucle 2
                        Condition pour les 8 direction possible deT
                        /*En fonction de la valeur NBaleatoire et du placement de R
                        principalement*/
                        {
                                Déplacement ou non de T ;
                                Ajout ou non de score en fonction du type d'herbe ;

                                Attribution d'une valeur à dejaFait ;
                                Blocage si R déjà présent ;
                                On sort de la boucle ;
                        }
                } //fin boucle 2
                        Condition pour dejaFait afin d'éviter répétition
                        {
                                dejaFait est remis à 0 ;
                                On sort de la boucle ;
                        }
        } //fin boucle 1
```

T dépend entièrement de la valeur de NBaleatoire détaillé ici, en effet elle indiquera la direction à prendre en fonction de la valeur retourné.
Le code se résume donc à une liste condition contenant l'application d'une direction, qui attendent la bonne valeur pour agir.
Voici les directions à prendre en fonctions des valeurs apportées :

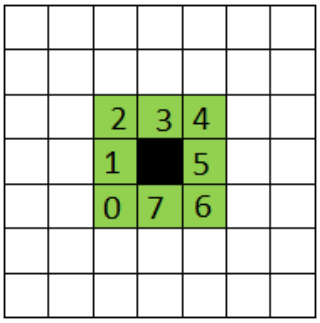


Image 3 : Grille de déplacement en fonction des chiffres entrés

Avec ce système de déplacement unique on peut simuler une trajectoire aléatoire sur une grille.

8.2- Comment évolue le temps T avec la tondeuse T ?

Puisque T est un robot à la direction imprévisible, le Temps T est condamné à ne pouvoir disposer d'une valeurs stable, d'une simulation à l'autre :

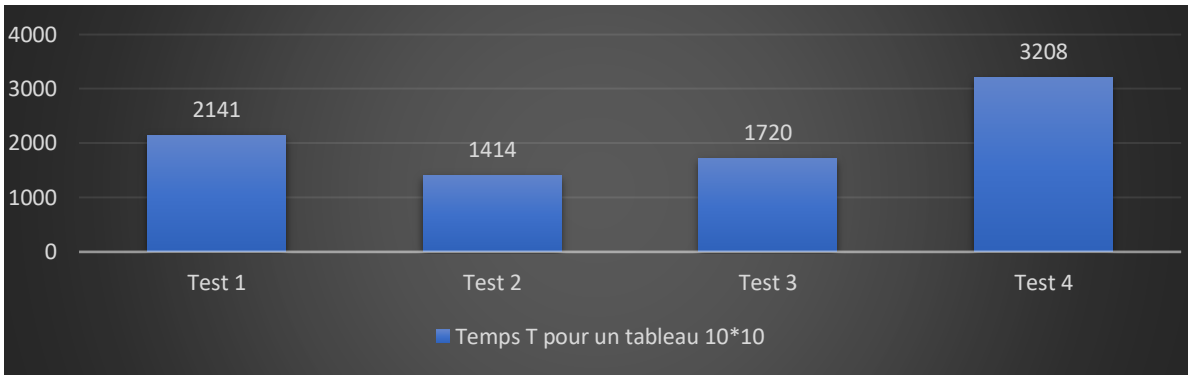


Figure 4 : Histogramme de comparaisons des scores en temps T des simulations aléatoire

La rapidité de T et la valeur en Temps se base donc sur la chance.

9.1- Etude de la grille et de son implication dans les performances :

Dans ce code, la taille de la grille est personnalisable, y a-t-il une corrélation entre les performances individuelles/mutuelles des tondeuses et du score de temps, avec la grille ?

9.2- Etude de la grille avec R :

On à vue précédemment que R sans le robot T obtenais, étant donné sa nature ordonné, toujours le même score qui est égal au T de temps, ou plus précisément :

Le score et le temp de R avec une grille interchangeable est égale à l'air de la grille, qui se définit par la multiplication du nombre de ligne par le nombre de colonne :

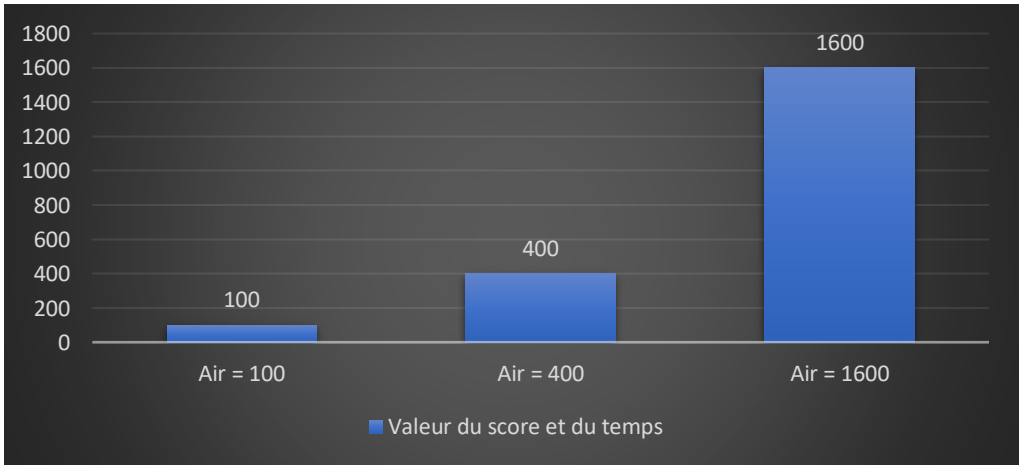


Figure 5 : Histogramme des résultat de R avec une grille personnalisable

9.3- Etude de la grille avec T :

Le robot tondeuse T ayant été programmé pour suivre une direction aléatoire, il rend son étude « chaotique ».
En effet les résultats varient d'un tirage à l'autre, mais la plus grande différence se présente lorsque l'on modifie la taille de la grille.
Quand on en vient à augmenter cette dernière et donc l'air total, la performance de T en temps chute drastiquement :

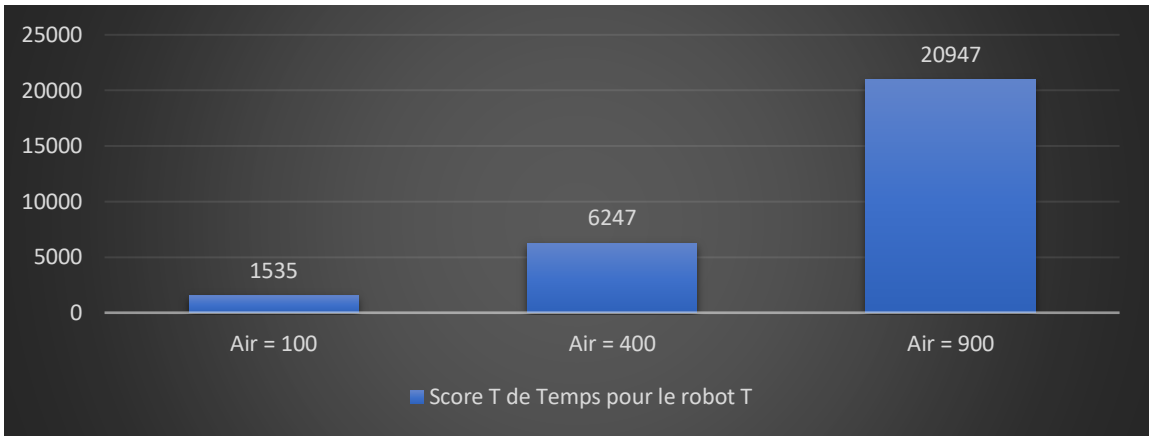


Figure 6: Histogramme des moyennes arrondie de T avec une grille personnalisable sur 2 tirages

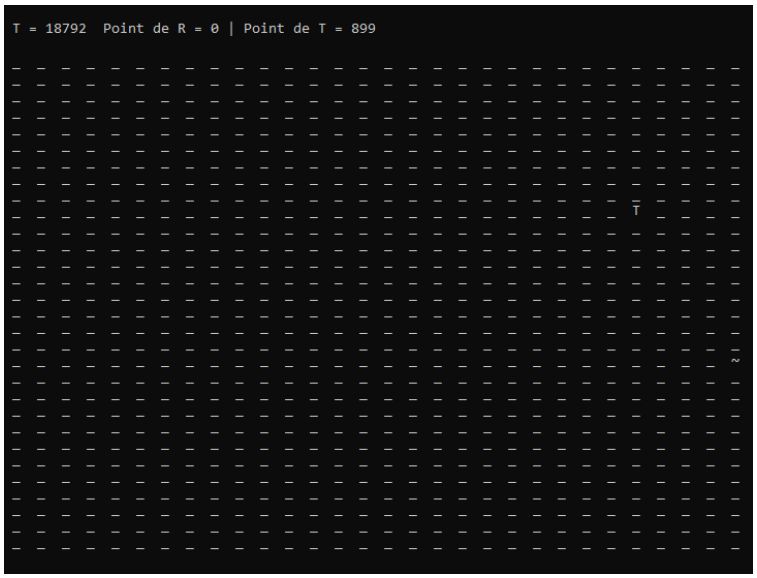


Image 4 : Le robot T ayant du mal à avoir la dernière case

Si une petite valeurs de temps est le résultat attendu pour une tondeuse performante, alors le robot T est surement le pire choix possible.

Le robot T à mis presque 5000 tour pour atteindre la dernière passerelle d'herbe sur cette simulation d'une grille ayant une air à 900.

9.4- Etude de la grille avec R et T :

Pour finir étudions le développement des score en simulation en fonction de la taille du tableau avec R et T fonctionnelle sur 5 tirages :

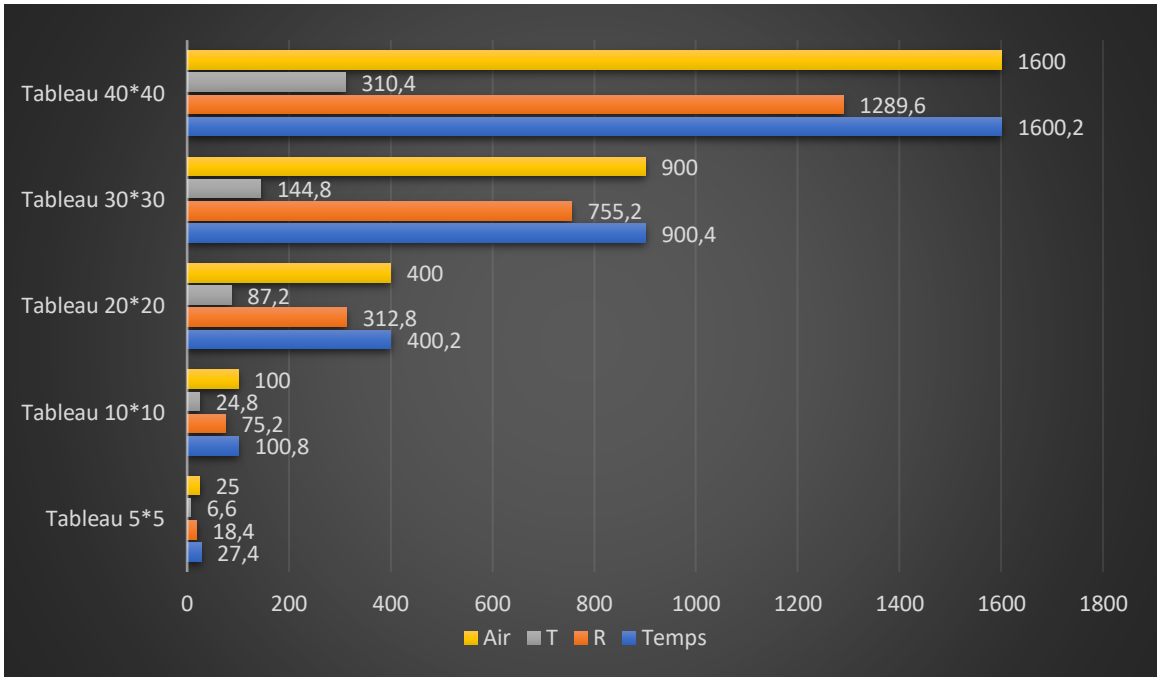


Figure 7 : Histogramme des moyennes de T, R , et de Temps avec un terrain de plus en plus grand représenté par l'Air.

Comme le montre les résultats, à première vue, le score de temp semble impacté positivement par l'augmentation de l'air du plan, ce qui pourrais s'expliquer par la chance diminuante pour R et T de se croiser ou de se bloquer dans un plan toujours plus grand .
La distance des deux entités augmente et les chances pour T de se diriger vers la direction R sont quant à elle fortement réduite, c'est pour cette raison qu'à partir du tableau 20*20 la décimale du Temps reste inférieur à 8.

Lors du tirage cependant une corrélation a pu être trouver :

Plus l'air du plan augmente plus le score de temps ne peut lui être inférieur, malgré les nombreuses simulations, on se rapproche peu à peu du score initiale de R étudié précédemment.
Cette démonstration signifie malheureusement, que la performance des deux tondeuses ne peut être amélioré en position initiale à partir d'une certaine taille, T devenant inutile à l'amélioration du score.

Hypothèse : T semble alors améliorer le score de temps uniquement en position final de R, ou quand il s'en approche, suppriment les dernières herbes de R et diminuant ainsi le score de temps :

*Comparons le score moyens sur 5 tirages de Temps du dernier graphique sur un tableau de 20*20 avec celui de l'hypothèse (toujours en 5 tirages).*

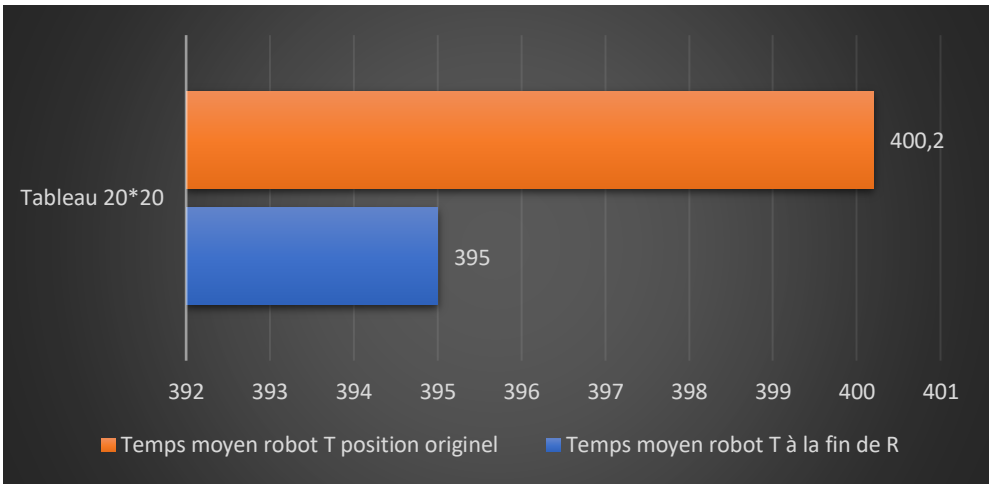


Figure 8 : Histogramme des moyennes de Temps normal et de Temps hypothèse

Grace au graphique l'hypothèse se confirme, en comparent les résultats du temps moyen sur 5 tirages, il semblerait que T en position final de R permette d'améliorer les performances de tonde et de descendre en dessous du score de R seul .
Comme on peut le voir sur le graphique, T en position originel à un temps supérieur de presque 5 point faisant de cette méthode, la moins efficace des deux.

Même si cela peut paraitre incohérent par sa contrainte de mouvement, Pour gagner du temps, il est donc plus profitable de placer T au coin de destination final de R, que de le placer partout ailleurs sur le terrain. Cette affirmation est d'autant plus vraie avec l'augmentation de l'Air du plan.