

A thick dark blue vertical bar runs down the left side of the page. A blue arrow-shaped box points to the right from this bar, containing the date. Below the bar, several thin, curved lines in dark blue and light grey sweep upwards and to the right.

27/03/2017

# Projet Intelligence Artificielle

Marie GUIRAUTE - Grégoire MELIN – Alexandre SENAUX

BORDEAUX INP- ENSC 2A

# TABLE DES MATIERES

Introduction	3
Avant-propos	3
Positionnement & objectifs	3
Equipe projet	3
Partie I : Les Chariots	4
1. Validation des exigences	4
2. Modélisation du projet	4
3. Architecture de la solution	4
4. Question 1 & 2	5
4.1. Justification des choix de conception	5
4.1.1. Liaison entre les questions 1 & 2	5
4.1.2. Heuristique utilisée	5
4.2. Description des états	6
4.3. Description de la recherche des successeurs	6
4.4. Calcul des coûts de déplacement	7
4.5. Réalisation	8
4.6. Comparaison avec et sans heuristique	8
4.7. Illustration du système par un cas d'utilisation	9
4.8. Illustration du système par des cas spécifiques	11
5. Question 3	13
5.1. Description des états	13
5.2. Explication de l'heuristique	13
5.3. Description d'une session	14
5.4. Les cas particuliers	15
Partie II : Réseaux de neurones artificielles	17
1. Perceptron une couche : Séparation linéaire de données	17
1.1. Modélisation	17
1.1.1. Modélisation de la solution	17
1.1.2. Schéma-bloc de l'algorithme	17
1.1.3. Représentation graphique des données d'entrées	18
1.1.4. Architecture du projet	19
1.2. Réalisation	19

1.2.1.	Justification de conception	19
1.2.2.	Illustration par un cas d'utilisation	19
2.	Perceptron multicouche	20
2.1.	Présentation de la solution	20
2.1.1.	Démarche	20
2.1.2.	Architecture de la solution	21
2.1.3.	Illustration par un cas d'utilisation	21
2.1.4.	Etude d'efficacité du réseau	23
2.1.4.1	Variations du coefficient d'apprentissage	23
2.1.4.1.	Choix du nombre d'itération	25
2.1.4.2.	Conclusion	26
3.	Classification	27
3.1.	Apprentissage Supervisé	27
3.1.1.	Réalisation	27
3.1.2.	Résultats	29
3.2.	Apprentissage Non-Supervisé	31
3.2.1.	Réalisation	31
3.2.2.	Résultats	31
3.	Gestion de Projet	34
3.1.	Répartition des tâches	34
3.2.	Planification	34
4.	Conclusion	35
4.1.	Apports	35
4.2.	Points Positifs	35
4.3.	Difficultés rencontrées	35
4.3.1.	Github et le « Merging »	35
4.3.2.	Clarté du code par défaut	35

# INTRODUCTION

## AVANT-PROPOS

Le projet est fait dans le cadre du module d'Intelligence Artificielle de Bordeaux INP-Ecole Nationale Supérieure de Cognitique.

## POSITIONNEMENT & OBJECTIFS

Le projet a pour but d'appliquer les notions vues en cours à savoir pour la partie I, les algorithmes d'exploration avec heuristiques, et pour la partie II, les différentes modalités d'apprentissage des réseaux de neurones artificiels à travers l'implémentation d'un perceptron pour la classification supervisée de données séparables linéairement d'une part, et d'autre part la création et l'utilisation de réseaux de neurones avec une technique d'apprentissage supervisée et non supervisée via l'implémentation d'un perceptron multi-couche.

## EQUIPE PROJET

Le projet a été assuré par Marie GUIRAUTE, Grégoire MELIN & Alexandre SENAUX, tous trois élèves de deuxième année à Bordeaux INP-Ecole Nationale Supérieure de Cognitique.

# PARTIE I : LES CHARIOTS

## 1. VALIDATION DES EXIGENCES

Tous les choix de conception et de modélisation ont été discutés et validés par le professeur encadrant des TD. A la fin de chaque question codée, nous avons donc montrés les résultats graphiques pour avoir une approbation ou une piste d'amélioration sur la façon d'optimiser le déplacement des chariots. C'est de cette façon que nous avons implémenté des conditions et des recherches nouvelles comme une création aléatoire des colis et une affectation des colis aux chariots en fonction de leur proximité.

## 2. MODELISATION DU PROJET

Pour avoir tous les cas nécessaires aux multiples déplacements des chariots et à la bonne livraison des colis, nous avons dû faire une modélisation de tous les cas possibles. Le chariot doit pouvoir se déplacer dans l'entrepôt, il a donc fallu faire une différence entre les allées et les étagères. Nous avons donc implémenté de nouvelles classes, « Chariot », « Colis » et « Emplacement » qui est une classe Fille de « GenericNode ».

Dans le « Program.cs », nous mettons deux variables : un entrepôt (une instance de la classe Entrepot) et un tableau de chariot. Ces deux variables sont utilisées pour les trois questions. Nous utilisons un troisième formulaire pour détailler les caractéristiques demandées, comme les nombres de nœuds ouverts et fermés, ou même l'arbre de recherche, sans qu'elles encombrent l'interface principal.

## 3. ARCHITECTURE DE LA SOLUTION

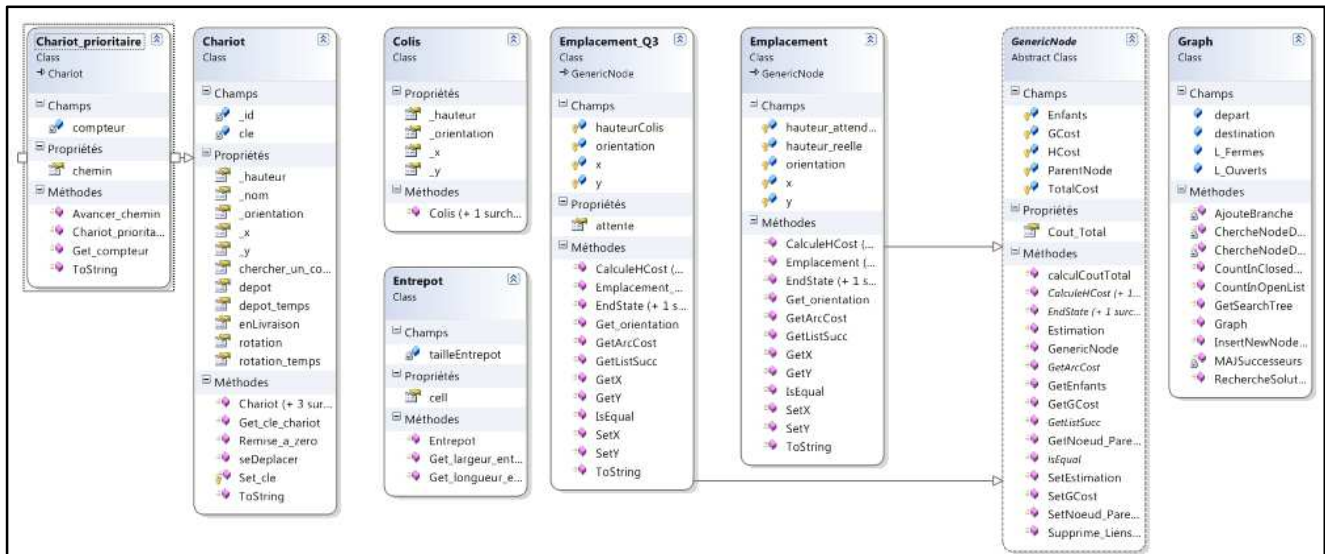
La question 1 a été réalisée dans la classe « Entrepot », nous avons conçu un entrepôt dont la taille et la largeur peuvent être variable. Les étagères ont une forme fixe comme sur le modèle donné dans le sujet. Les deux colonnes et les deux lignes des extrémités sont toujours laissées libres pour les circulations des chariots, comme sur l'exemple montré dans le sujet. La classe « Entrepot » possède donc un tableau nommé « cell », avec des entiers qui est une représentation de l'entrepôt à un instant  $t$  (0 pour les cases libres, -1 pour des cases avec des étagères, 2 pour des cases avec des chariots) et les caractéristiques (la taille et la largeur) de l'entrepôt, contenues dans un second tableau d'entiers.

Pour les colis, il a donc été nécessaire de créer une classe Colis avec une position dans l'entrepôt ( $x,y$ ) avec une orientation sous forme d'une string et une hauteur soit un entier entre 0 et 10. Il existe différents constructeur de colis : un constructeur où on donne toutes les valeurs des attributs pour la question 2, et un constructeur où toutes les valeurs sont décidées de manière aléatoire pour la question 3. Dans la question 2, c'est l'utilisateur qui décide de la position du colis (comme indiqué dans le sujet), alors que pour la question 3, les colis sont définis et placés sans l'intervention de l'utilisateur.

Les chariots disposent de multiples attributs nécessaires à leur fonctionnement, comme des booléens pour

indiquer si le chariot est en rotation, ou en train de prendre son colis ou en retour vers la plateforme de livraison. Des compteurs entiers ont été mis en place et sont incrémentés lors des différents déplacements des chariots pour faire une mesure du temps qu'il passe en rotation par exemple.

La classe « GenericNode » qui a été donnée par les professeurs encadrant les TD possède deux classes filles car nous avons créé deux classes : « Emplacement » pour la question 2 et « Emplacement\_Q3 » pour la question 3, sur les conseils de Mr Salotti. Les deux classes possèdent des codes similaires, mais la classe « Emplacement\_Q3 » permet de faire une évaluation temporelle des chemins et des meilleurs déplacements possibles. Avec cette nouvelle classe, nous avons pu faire une évaluation des chemins en tenant compte de l'indisponibilité temporelle des cases.



## 4. QUESTION 1 & 2

### 4.1. JUSTIFICATION DES CHOIX DE CONCEPTION

#### 4.1.1. LIAISON ENTRE LES QUESTIONS 1 & 2

La question 1 et la question 2 sont traitées dans la même partie car elles sont très liées, l'une étant la suite de l'autre.

#### 4.1.2. HEURISTIQUE UTILISEE

Pour faire l'heuristique, nous avons utilisé la distance de Manhattan car le déplacement du chariot pouvait se baser sur une telle heuristique : il navigue entre les étagères et les autres chariots comme un taxi dans les blocs de Manhattan. Nous avons donc pris une heuristique simple et appropriée.

La distance de Manhattan est égale à la somme des valeurs absolues des différences de coordonnées en X et en Y. La mesure de la distance de Manhattan est plus grande (ou égale) que celle de la distance Euclidienne,

ce qui nous a paru une meilleure approximation pour ce projet.

#### 4.2. DESCRIPTION DES ETATS

Pour les états, nous avons utilisé, dans la question 1, une définition des emplacements avec x et y pour lui donner une place dans l'entrepôt.

Pour la question 2, nous avons, en plus, rajouté l'orientation car il fallait pouvoir différencier deux états de même localisation dans l'entrepôt, mais avec une orientation différente.

Nous avons, par la suite, aussi ajouté une caractéristique à cet état pour la hauteur que doit atteindre le chariot lorsqu'il va chercher son colis. Cet ajout a permis de prendre en compte le temps que met le chariot pour monter son bras métallique à la bonne hauteur et ainsi de bien arriver à sa position finale. On différencie donc les deux Emplacements entre deux chariots quelconques par leur position (x,y), leur orientation et la hauteur de leur bras robotique.

#### 4.3. DESCRIPTION DE LA RECHERCHE DES SUCCESEURS

Pour trouver les successeurs d'un nœud, nous avons dû implémenter la méthode `public override List<GenericNode> GetListSucc()` provenant de la classe mère `GenericNode`. Nous avons différencié 3 situations :

- Les 4 angles de l'entrepôt : ces 4 points spécifiques ne disposent que de deux voisins, car on ne compte pas les déplacements en diagonale. La modélisation ci-contre est faite pour un tableau de 25 par 25, mais le code fonctionne pour toutes les tailles d'entrepôt. Il s'agit des cases en rouge sur le schéma de l'entrepôt ci-dessous.

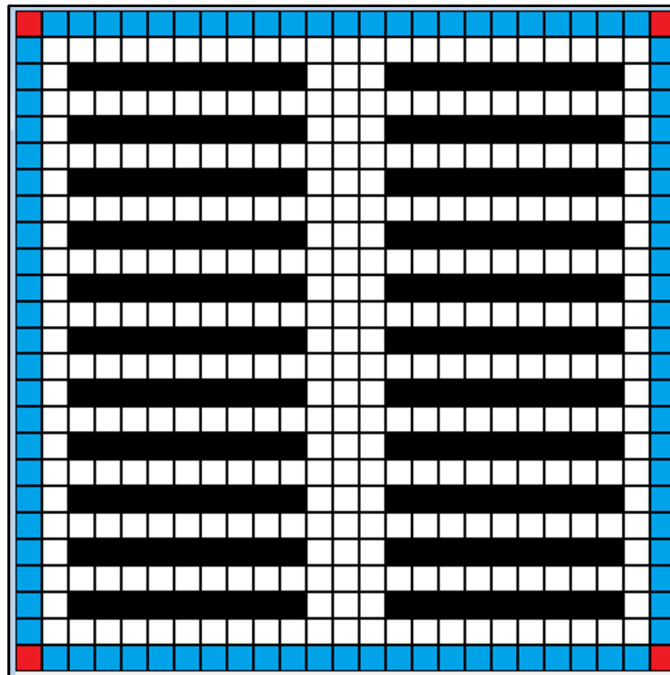
```
#region Definition des 2 voisins : 4 angles
/*MODELISATION :
    [0,0]  0,1    ...    23,0  [24,0]
      1,0          24,1
      ...          ...
      ...          ...
    0,23          24,23
    [0,24] 1,24    ...    23,24 [24,24]
*/
```

- Les cases sur les bords de l'entrepôt en dehors des angles : ces cases n'ont que trois voisins. Nous avons donc fait des conditions pour vérifier si l'emplacement du chariot (la case du tableau) est sur les bords. Cela se traduit par un x ou un y qui vaut 0 ou encore par un x ou un y qui vaut la longueur ou la largeur de l'entrepôt -1 à cause des indices du tableau qui commence à 0. Il s'agit des cases en bleu sur le schéma.

```
/* MODELISATION :
    ... i,j-1 [i,j] i,j+1 ...
      .          .
    ig-1,j      i+1,j      a-1,b
    [ig,jg] ig,jg+1      a,b-1 [a,b]
    ig+1,jg      a+1,b
      .          .
      ... c-1,e      c,e-1 [c,e] c,e+1 ...
*/
```

- Les cases avec 4 voisins théoriques : ce sont toutes les autres cases du tableau. Il n'est pas possible que les cases correspondant à des étagères soient testées par cette fonction car nous avons mis une

sécurité en amont. Pour ces cases blanches, elles possèdent donc les 4 voisins mais on évalue si la case adjacente est un autre chariot (représenté par un 2) ou bien une étagère (représenté par un -1). Et dans le cas contraire, on l'ajoute dans la liste des successeurs de ce nœud.



On dispose donc de 2 à 4 nœuds successeurs en fonction des cas.

#### 4.4. CALCUL DES COÛTS DE DEPLACEMENT

Pour les coûts de déplacement, nous avons choisi de faire un coût pour le passage d'une case à une autre. Par exemple, lorsque le chariot souhaite aller d'une case à une autre, mais qu'il doit faire une rotation pour cela, on compte un coût de 4 : soit 3 secondes pour la rotation et 1 pour changer de case.

Pour savoir si le déplacement nécessite une rotation, nous devons donc regarder le placement entre les x et y des deux emplacements, mais aussi l'orientation car elle va déterminer si on doit faire une rotation.

Le dernier et troisième coût est le coût final : le temps que met le robot pour prendre son colis et pour être prêt à pouvoir repartir. Nous avons donc compté deux secondes par mètre, puis 10 secondes pour que le bras robot prenne le colis et encore 2 secondes par mètre pour redescendre à 0 mètre de hauteur. Pour savoir si l'emplacement que nous analysons est l'emplacement final, nous utilisons une autre fonction EndState() qui permet de savoir si cet emplacement est l'emplacement final avec la hauteur, l'orientation et la position (x,y).

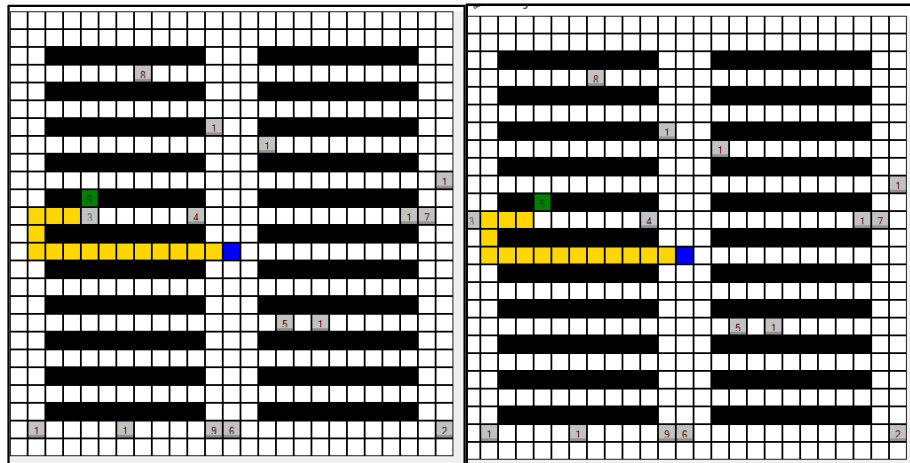
```
if (EndState(succ))
{
    return (4 * hauteur_attendue + 10);
}
else
{
    if (orientation == "Nord")
    {
        if (succ.y == this.y - 1) { return (1); }
        else { return (4); }
    }
    else if (orientation == "Sud")
    {
        if (succ.y == this.y + 1) { return (1); }
        else { return (4); }
    }
    else if (orientation == "Est")
    {
        if (succ.x == this.x + 1) { return (1); }
        else { return (4); }
    }
    else if (orientation == "Ouest")
    {
        if (succ.x == this.x - 1) { return (1); }
        else { return (4); }
    }
}
```



#### 4.5. REALISATION

Dans l'image ci-dessous, nous avons fait un déplacement entre le point de départ (la case bleue) et la position finale qui se trouve en dessous du colis car le colis est orienté au Sud. Le chariot est en gris pendant son déplacement, il fait donc le tour de l'étagère car un autre chariot (qui est statique dans cette question) bloque l'accès au colis par l'Ouest.

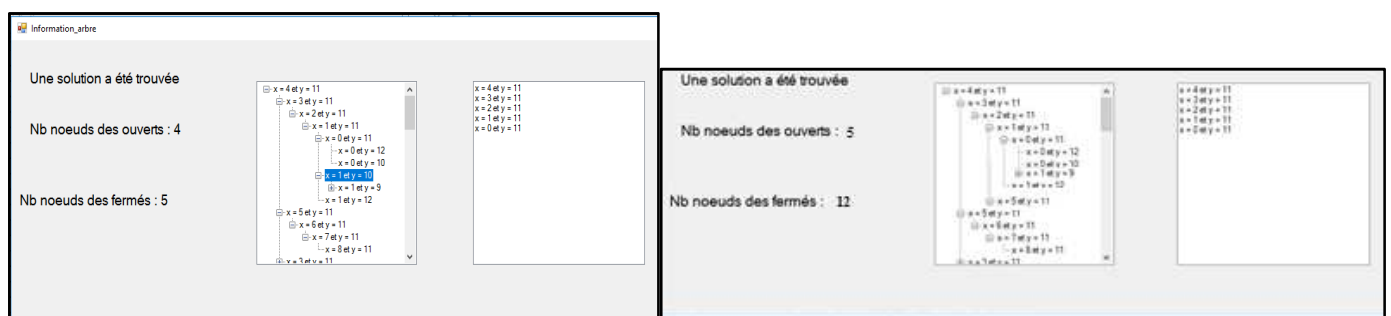
Le chariot fait ensuite son chargement et s'arrête. Dans le formulaire, vous disposez d'un bouton « Détails » qui ouvrira de façon modale une nouvelle fenêtre, en vous montrant le nombre de nœuds ouverts et fermés ainsi que l'arbre de résolution. Vous décidez ensuite d'envoyer le chariot déposer son colis et vous obtenez l'affichage de droite : le chariot est revenu sur la colonne 1 qui est la plateforme de livraison. Il a bien choisi le plus court chemin pour revenir à la plateforme de livraison.



Si, au retour du chariot, on souhaite regarder les détails, on peut avoir une nouvelle fenêtre.

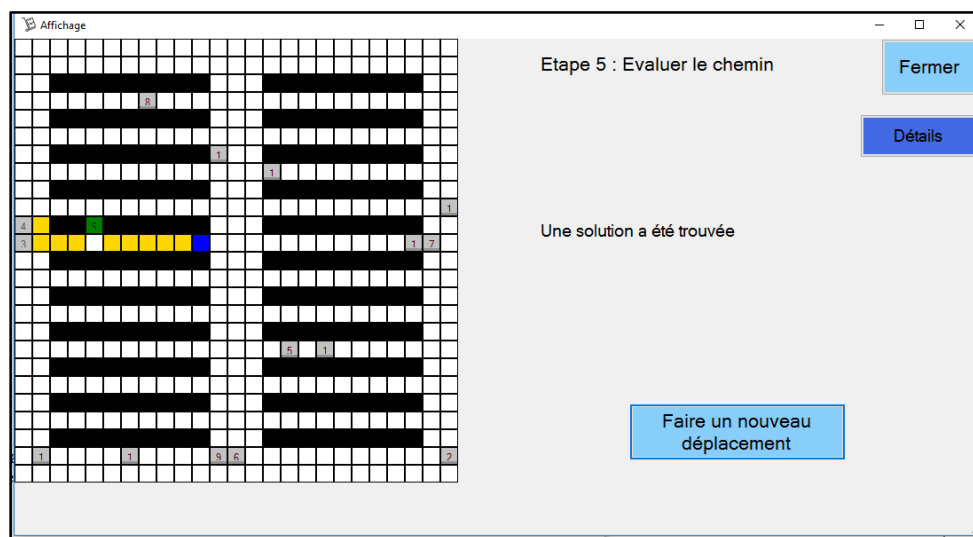
#### 4.6. COMPARAISON AVEC ET SANS HEURISTIQUE

L'image ci-dessous de droite est celle utilisant une heuristique et l'image de gauche est celle sans heuristique. Nous constatons donc que le nombre de nœuds fermés et le nombre de nœuds ouverts augmentent quand on n'utilise pas d'heuristiques. Notre heuristique a donc une relative efficacité.



Si on fait déplacer un nouveau chariot pour prendre un colis, au même endroit, la position finale pour déposer le colis (celle sur la colonne 1) ne peut plus être la case en face de l'étagère car il y a déjà un chariot qui est déjà en dépôt sur cette ligne. Le chariot 2 va donc trouver un nouvel emplacement sur la colonne 1 qui doit être le plus près possible de son emplacement initial. Pour le plus court chemin, il s'agit donc de l'emplacement

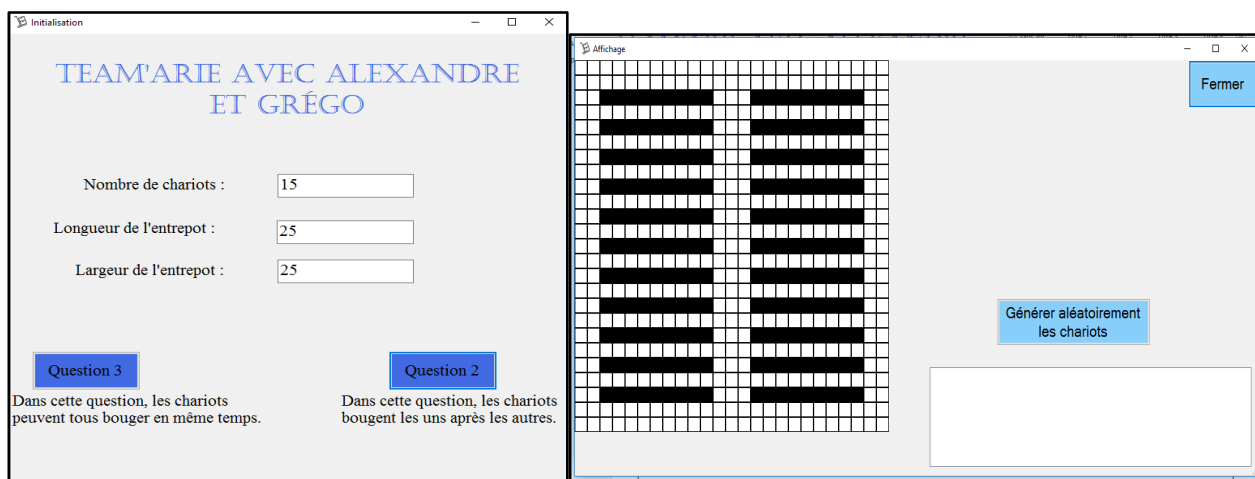
au-dessus ou en dessous de l'emplacement du chariot 1.



Pour faire un nouveau déplacement, vous devez cliquer sur le bouton et ensuite choisir votre chariot et placer un nouveau colis. Nous avons mis une sécurité sur le choix des chariots et sur le placement du colis pour éviter les erreurs ou les exceptions non gérées et donc un bug du programme.

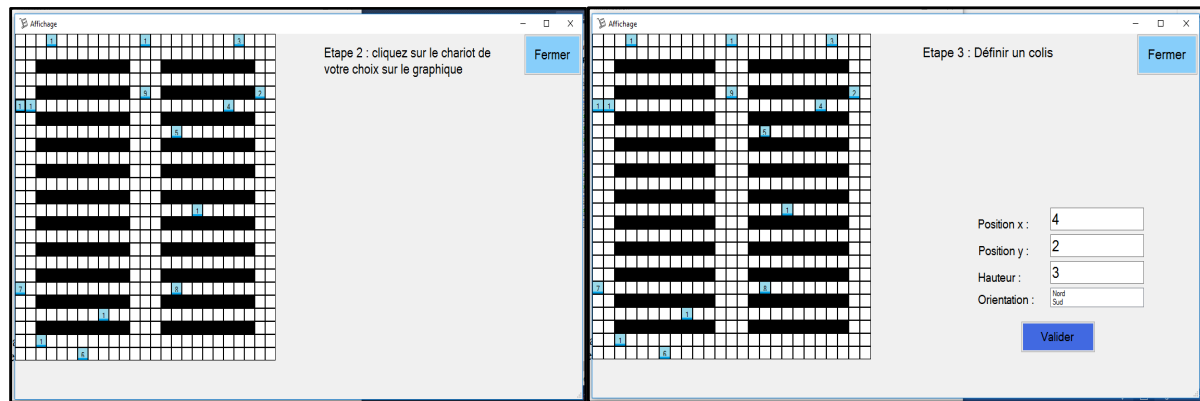
#### 4.7. ILLUSTRATION DU SYSTEME PAR UN CAS D'UTILISATION

Lorsque vous lancez le projet sur les chariots, vous arriverez sur une page intermédiaire vous demandant de préciser combien de chariots vous souhaitez, ainsi que les dimensions de l'entrepôt (longueur et largeur). La longueur de l'entrepôt correspondrait à l'axe des abscisses et la largeur de l'entrepôt à l'axe des ordonnées. L'entrepôt a donc une taille variable et les déplacements des chariots marchent pour différentes tailles d'entrepôt.

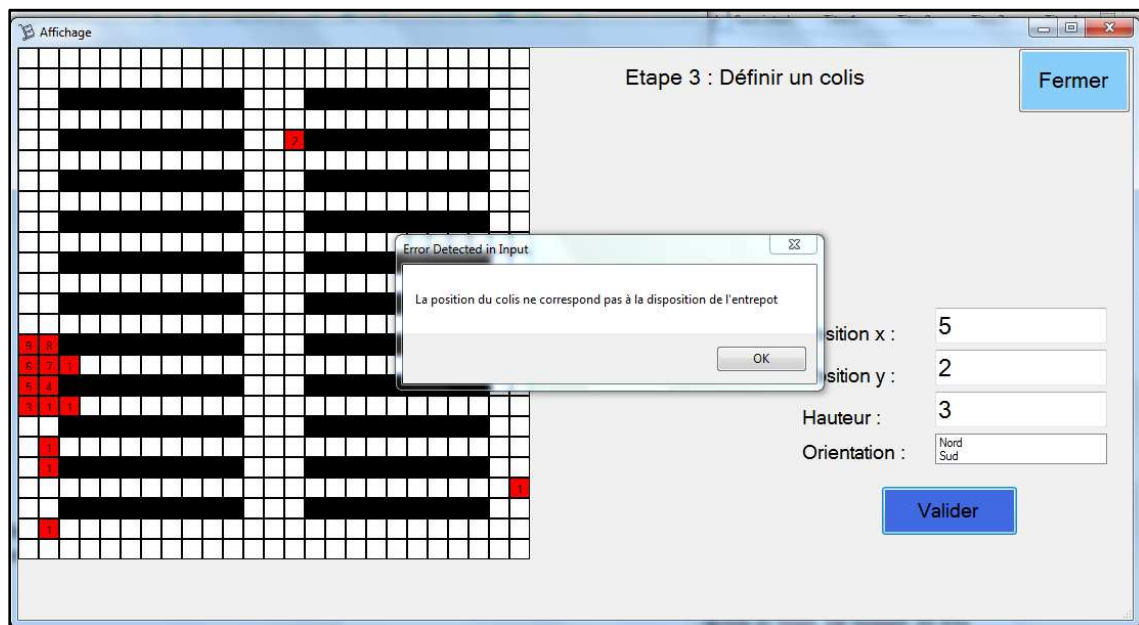


Une fois que ces paramètres sont selon votre souhait, vous devez valider votre réponse en cliquant sur le bouton de droite avec écrit "Question 2". Des paramètres sont mis par défaut, libre à vous de les changer ou de les garder. Après avoir validé et choisi quel type de question vous souhaitez tester, vous devez placer les chariots. Il existe donc deux façons : une façon rapide avec un placement aléatoire et un placement manuel où vous cliquez sur les cases blanches du tableau pour positionner vos chariots. Une fois tous les chariots placés,

on vous demande de sélectionner un chariot parmi ceux qui viennent d’être définis en cliquant sur lui. Une fois la sélection faite, on vous propose de créer un colis que le chariot ira chercher.



Des valeurs par défaut sont également proposées, mais veillez à placer le colis au bon endroit. Si vous placez le colis en dehors des étagères (qui sont représentées en noirs sur le graphique), une fenêtre vous annonce qu’il y a une erreur dans votre saisie et que le colis ne peut pas être comptabilisé. Cette propriété marche pour toutes les tailles d’entrepôt. Les cases du tableau comme les tableaux commencent à 0 pour garder une cohérence avec le milieu où évoluent les chariots.



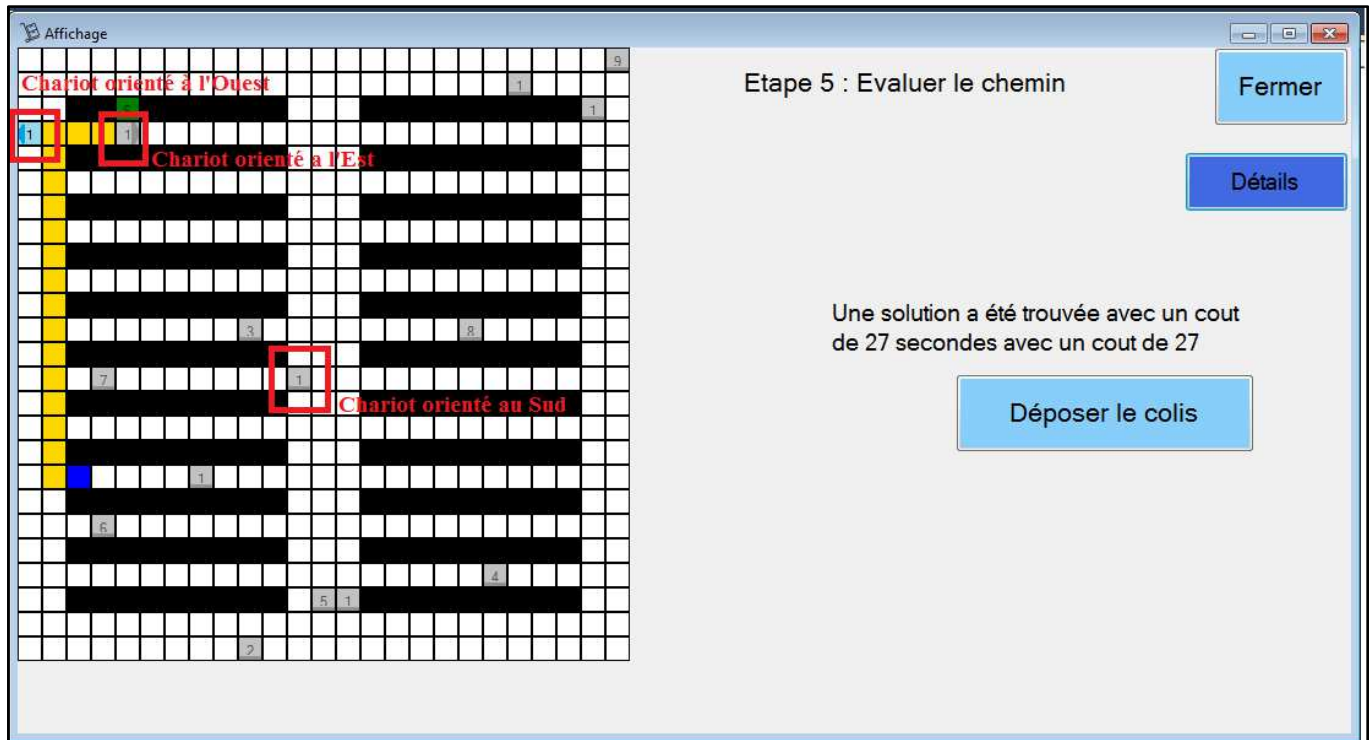
Une fois les quatre coordonnées du colis correctement remplies, il faut que vous cliquiez sur « Valider » pour que le chariot aille chercher le colis. Le déplacement du chariot se fait donc en fonction du temps, par exemple, les trois secondes de rotation sont respectées et le temps pour aller chercher un colis l’est aussi. Quand le chariot se déplace, les autres chariots sont statiques dans cette question et les boutons sont inactivés. La valeur du coût en termes de secondes est indiquée sur le côté de la fenêtre.

Si les chariots sont générés aléatoirement, vous pouvez voir grâce à une barre d’un bleu plus foncé l’orientation du chariot. Quand cette barre est vers le bas, c’est que le chariot est orienté au Sud. Les chariots ont une couleur différentes en fonction de leur type de placement : aléatoire ou au clic.

#### 4.8. ILLUSTRATION DU SYSTEME PAR DES CAS SPECIFIQUES

##### → Les orientations des chariots

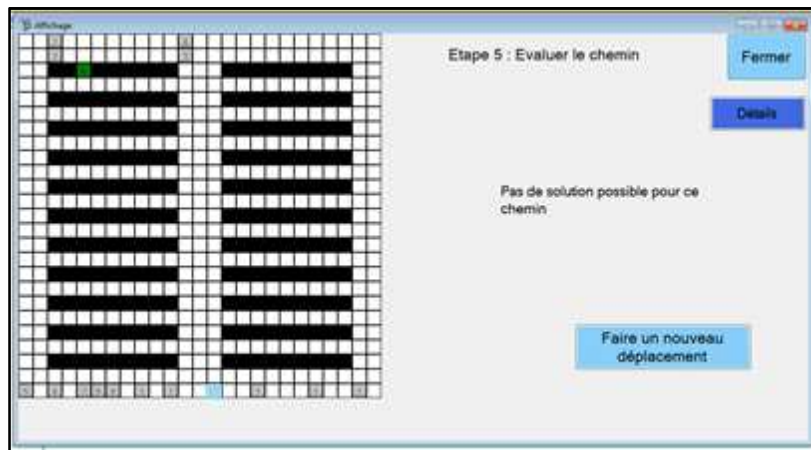
Nous avons choisi, dans cette question, de mettre des images différentes sur les boutons représentant le chariot dans son déplacement. Ces images permettent de comprendre et de visualiser le sens du chariot. Par exemple, dans l'image ci-dessous, nous avons des chariots à l'Est lorsque le chariot est à la plateforme de dépôt ou l'Ouest lorsque le chariot vient chercher son colis depuis l'Ouest. Tous les autres chariots sont au Sud.



##### → Pas de solution

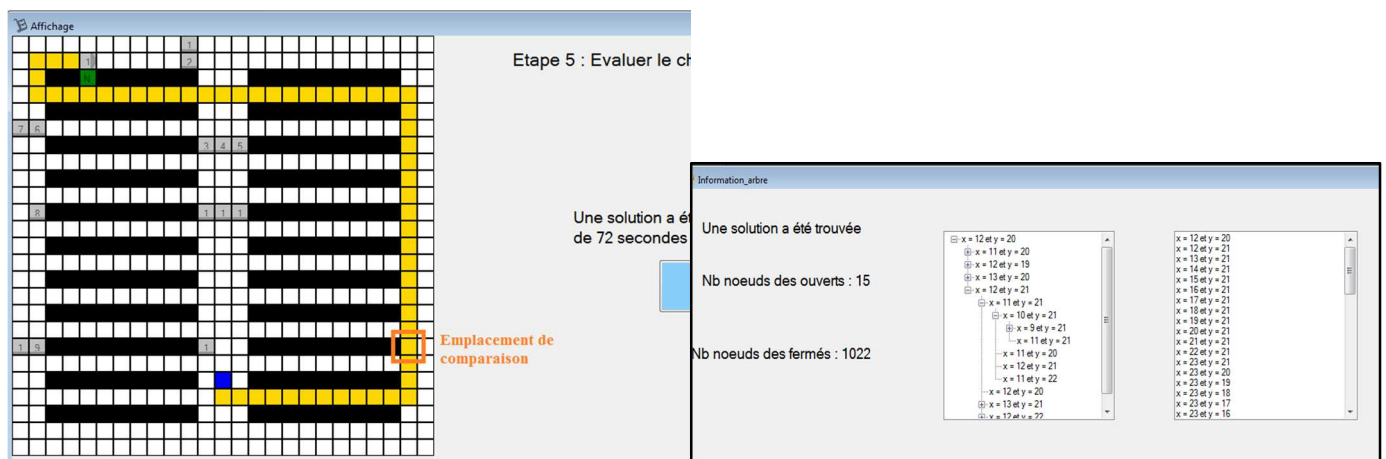
Quand il n'existe pas de chemin pour qu'un chariot aille chercher son colis, la non-solution est également affichée et on vous propose de recommencer un nouveau déplacement et donc de choisir un nouveau chariot puis un nouveau colis. Dans la configuration de l'exemple, les chariots du haut ont été placés de façon à bloquer l'accès aux autres chariots et donc le chariot choisi (ici en bleu) ne peut pas accéder à son colis (ici en vert et orienté au Nord).

Le chariot ne se déplace pas et reste à sa position initiale.



### → Un chemin difficile

Le chariot que nous venons de choisir est sur la case en bleu foncé dans l'entrepôt et il doit aller chercher le colis qui est en vert, orienté au Nord. Le chariot était orienté au Sud à son début. Pour aller chercher son colis, il doit donc éviter les autres chariots sournoisement placés. Le chariot, étant initialement orienté au Sud, le chemin est donc celui qui limite le nombre de rotation car elles demandent plus de temps qu'un déplacement en ligne droite ; c'est donc pour cela que le chariot fait son premier déplacement vers le Sud et non vers le Nord. Pour faire ce déplacement, on explore 1022 nœuds pour trouver le meilleur chemin.



En faisant ce mouvement vers le bas et non vers le haut, on gagne 1 seconde car le chemin dont le premier emplacement est vers le Haut demande 3 virages et 13 déplacements classiques soit 22 secondes alors que l'autre chemin ne demande que 2 virages pour 15 déplacements soit 21 secondes. Pour faire ces calculs, on compte comme fin quand les deux chemins se rejoignent avec un chariot dans une même position soit la case encadrée en orange.

## 5. QUESTION 3

### 5.1. DESCRIPTION DES ETATS

Pour cette question, sur les conseils des professeurs consultés, nous avons donc créé d'autres classes que les classes utilisées pour la question 2 car elles demandaient un fonctionnement plus spécifique et des attributs plus nombreux. Nous avons donc une classe « Emplacement\_q3 » et une classe « Chariot\_prioritaire ».

L'emplacement est désormais caractérisé par la position (x,y), l'orientation et la hauteur réelle. Ils ont permis de faire une comparaison plus fine entre deux emplacements lorsque du calcul des différents chemins en tenant compte des chariots avant lui. Des accesseurs ont été mis en place.

Les chariots prioritaires possèdent des attributs leur donnant des spécificités que n'ont pas les chariots classiques. Nous avons dû inclure dans les caractéristiques du chariot dans cette nouvelle spécificité. Le chariot prioritaire est le chariot que choisit la personne pour aller chercher directement son colis sans tenir des comptes des autres chariots qui eux tiennent compte du trajet du chariot prioritaire et des autres chariots.

```
public List<GenericNode> chemin { get; set; }
private int compteur ;

public void Avancer_chemin() { compteur++; }
public int Get_compteur() {return(compteur);}
|
// CONSTRUCTEURS
// utilisation du constructeur de la classe mère
public Chariot_prioritaire(int cle, int x,int y,string orientation,int hauteur): base (x
{
```

Les chariots prioritaires possèdent donc toutes les fonctionnalités des chariots classiques pour les déplacements, les rotations et la prise du colis.

### 5.2. EXPLICATION DE L'HEURISTIQUE

Pour faire une heuristique, nous avons donc dû compter les différents cas. Si l'emplacement final possède le même y que l'emplacement que l'on teste, il faut donc que le chariot se déplace en ligne droite jusqu'à son emplacement final, on ne compte donc que la distance entre les cases.

Si les deux emplacements ont le même x, les deux emplacements ont donc une ou plusieurs rangées qui les séparent. On compte donc une pondération du coût total en ligne droite par +9 secondes car il faut compter les trois virages que le chariot devra faire pour contourner les étagères de chacun trois secondes.

```

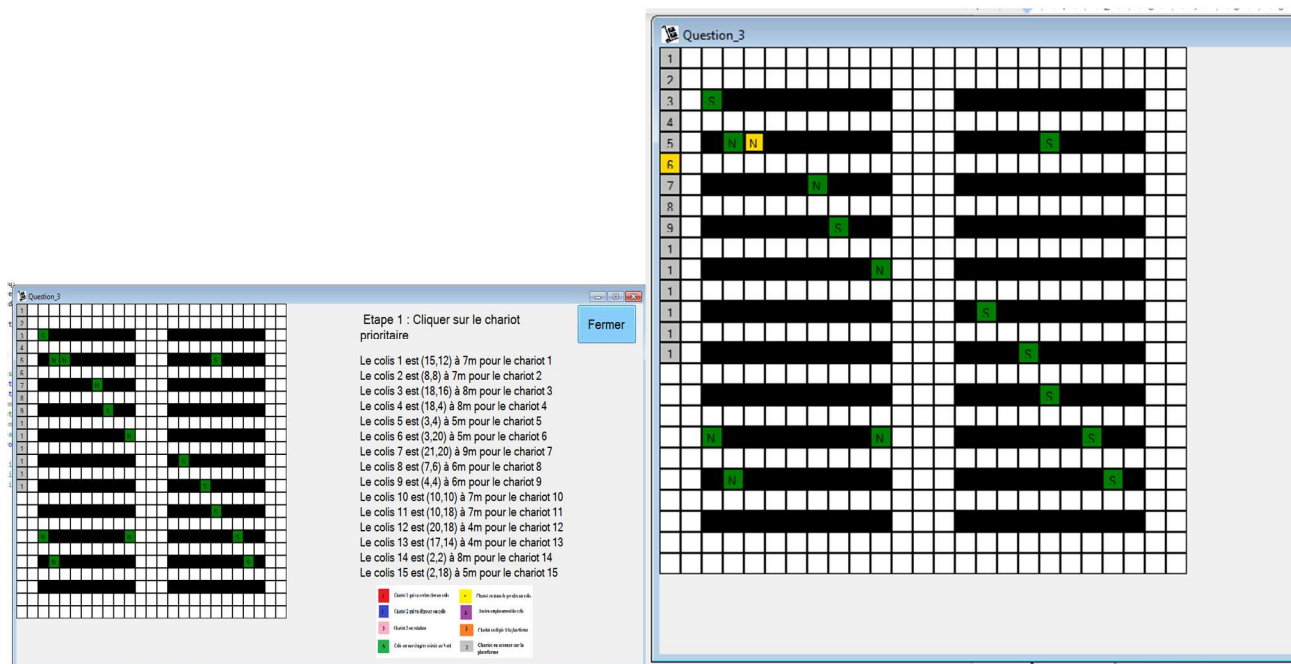
public override void CalculeHCost(Emplacement final)
{
    // distance de Manhattan
    if (this.y == final.y)
        this.HCost = HCost + (Math.Abs(x - final.x) + Math.Abs(y - final.y));
    else if (this.x == final.x)
        this.HCost = HCost + (Math.Abs(x - final.x) + Math.Abs(y - final.y)) + 9;
    else
        this.HCost = HCost + (Math.Abs(x - final.x) + Math.Abs(y - final.y)) + 6;
}

```

Si les deux emplacements ne correspondent ni en x ni en y, on doit donc faire un calcul sur la distance de Manhattan avec un +6 pour les deux virages minimal nécessaire à un déplacement sur un x et un y différents entre le point de départ et le point d'arrivée.

### 5.3. DESCRIPTION D'UNE SESSION

Quand on lance le programme, on a donc le premier affichage qui demande le nombre de chariots que vous voulez placer et la taille de l'entrepôt. On valide ensuite la réponse, des valeurs par défaut sont proposées si vous souhaitez les utiliser. Vous arrivez ensuite sur une fenêtre avec l'entrepôt et une légende sur le code couleur et les caractères utilisés est indiquée pour une meilleure compréhension des configurations du chariot. Le premier affichage de la question 3 est l'image de gauche.

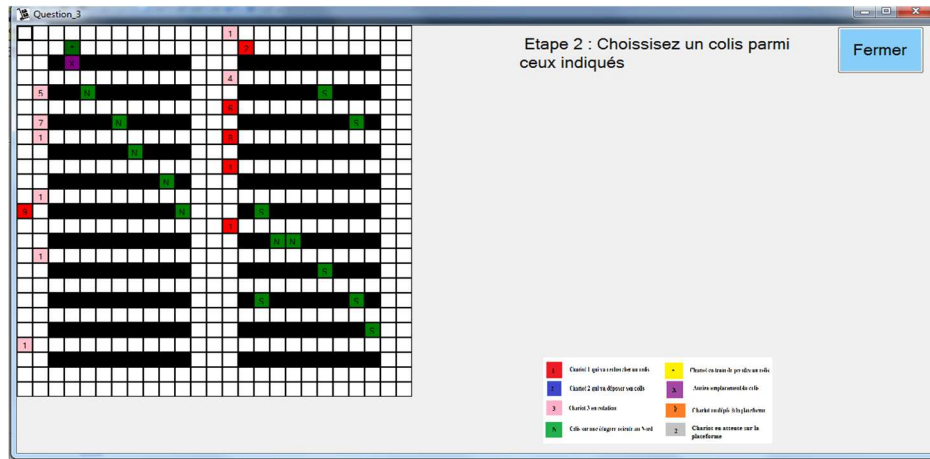


La description des colis avec le numéro des chariots qui va le chercher permet de vérifier que les chariots vont bien chercher les colis qui leur ont été attribués. Par défaut, il y a une répartition aléatoire des colis pour les chariots, mais quand vous indiquez qu'un chariot va chercher le colis choisi. Une répartition des chariots se fait en fonction de leur colis le plus prêt quand c'est possible et sinon on conserve l'affectation aléatoire. Quand vous avez cliqué sur le chariot et le colis affecté, la partie peut commencer et un bouton apparaît.

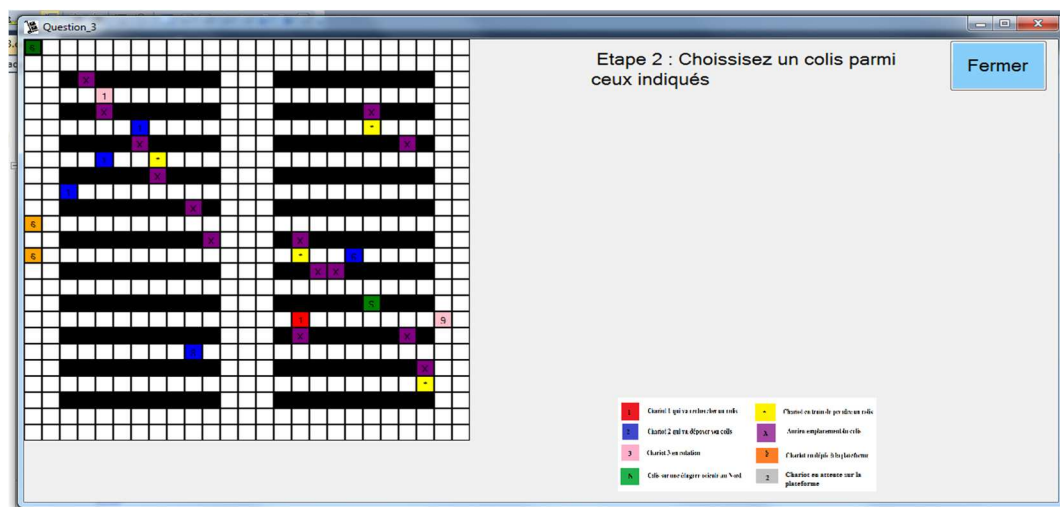


## 5.4. LES CAS PARTICULIERS

Les déplacements se faisant tous ensemble, il est difficile de faire des screens. Dans l'exemple ci-dessous, nous avons donc choisi le chariot en vert foncé comme chariot prioritaire. Il est en train de prendre son colis qui est le plus en haut à gauche.



Comme ce chariot est le chariot prioritaire, on a donc un chariot qui a un retour à la plateforme rapide. Le chariot prioritaire garde une couleur différente des autres pour qu'on puisse le suivre, mais les rotations ne sont pas différenciables du déplacement d'un point de vue de la couleur.



A la fin du programme, tous les chariots sont revenus à la plateforme selon le chemin le plus court depuis l'emplacement où ils ont pris leur colis. Lorsque tous les chariots sont placés, on affiche les caractéristiques de quel chariot va chercher quel colis. Théoriquement d'un point de vue du code, on demande d'afficher ce texte dès que la personne appuie sur « Valider », mais comme on joue sur le Thread pour faire un déplacement qui soit perceptible dans le temps par l'utilisateur, l'affichage ne se fait que lorsque les chariots sont revenus à la plateforme.



Question\_3

**Etape 2 : Choisissez un colis parmi ceux indiqués**

Le colis 1 est (19,16) à 4m pour le chariot 0  
 Le colis 2 est (19,4) à 4m pour le chariot 1  
 Le colis 3 est (3,2) à 1m pour le chariot 2  
 Le colis 4 est (10,12) à 1m pour le chariot 3  
 Le colis 5 est (7,8) à 2m pour le chariot 4  
 Le colis 6 est (17,14) à 2m pour le chariot 5  
 Le colis 7 est (15,12) à 3m pour le chariot 6  
 Le colis 8 est (15,18) à 3m pour le chariot 7  
 Le colis 9 est (21,18) à 0m pour le chariot 8  
 Le colis 10 est (21,6) à 0m pour le chariot 9  
 Le colis 11 est (6,6) à 7m pour le chariot 10  
 Le colis 12 est (4,4) à 1m pour le chariot 11  
 Le colis 13 est (9,10) à 1m pour le chariot 12  
 Le colis 14 est (16,14) à 2m pour le chariot 13  
 Le colis 15 est (22,20) à 6m pour le chariot 14

1	Chariot 1 qui va chercher un colis	-	Chariot ou train de perchoir au rail
2	Chariot 2 qui va déposer ses colis	X	Autres emplacements des colis
3	Chariot 3 en rotation	3	Chariot en dépôt à la fin de la course
X	Colis sur une étagère située au Nord	2	Chariot en attente sur la plateforme

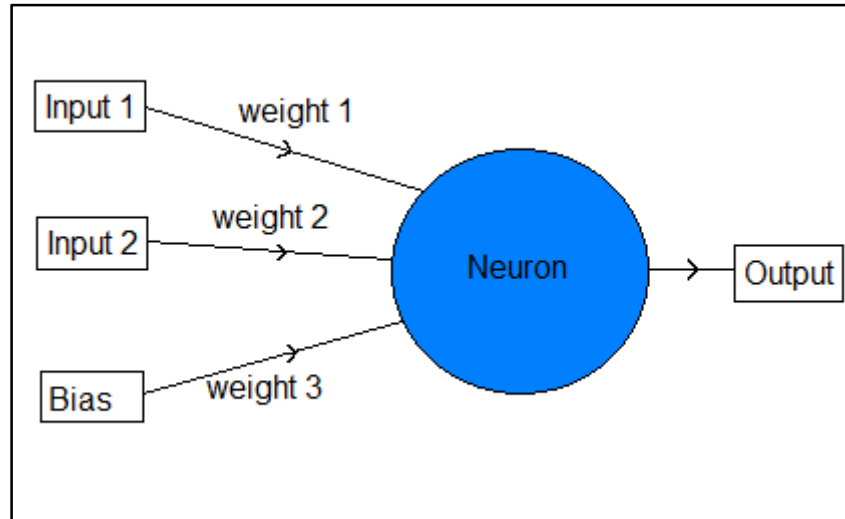
Fermer

# PARTIE II : RESEAUX DE NEURONES ARTIFICIELLES

## 1. PERCEPTRON UNE COUCHE : SEPARATION LINEAIRE DE DONNEES

### 1.1. MODELISATION

#### 1.1.1. MODELISATION DE LA SOLUTION

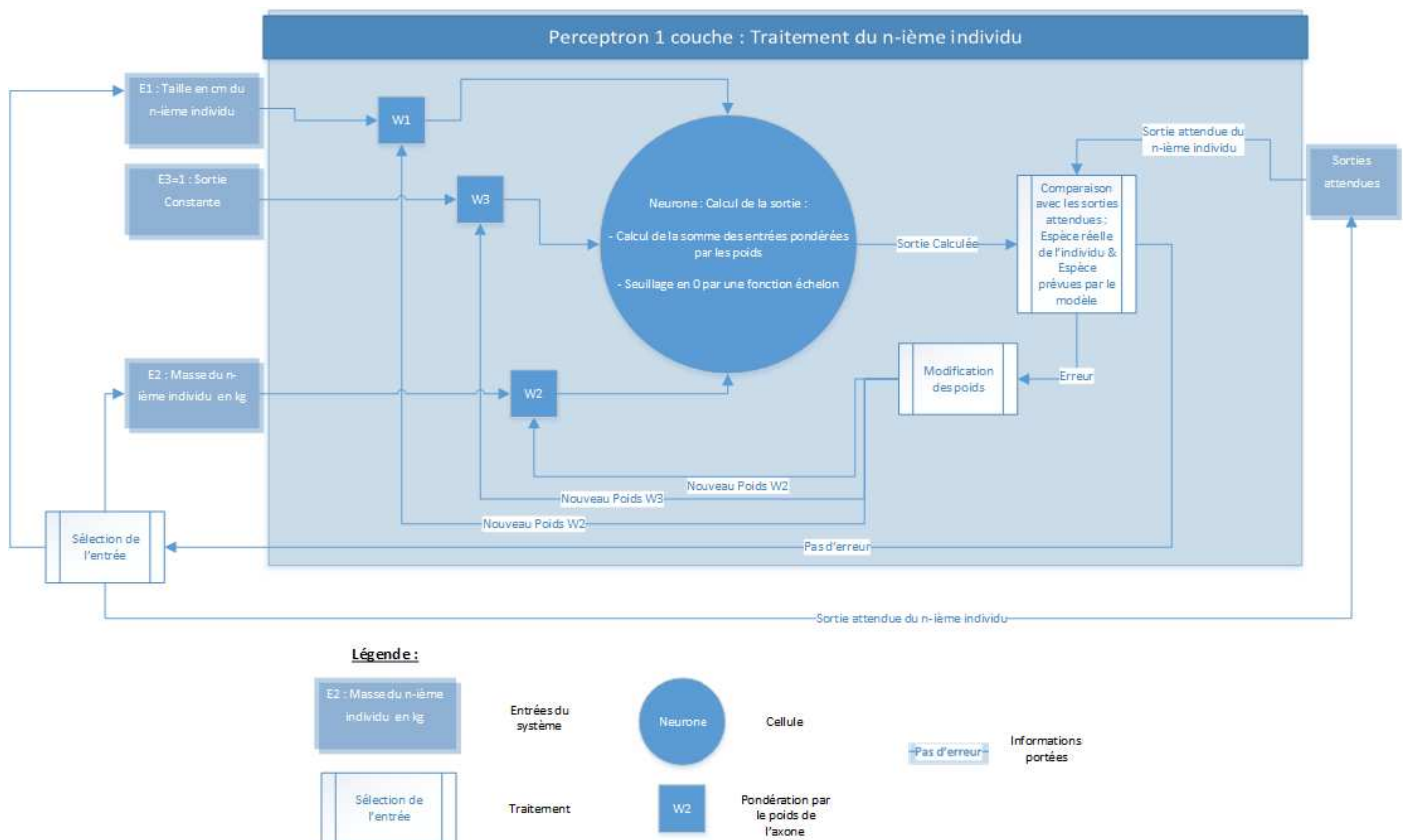


On se retrouve dans le cas basique du perceptron, c'est-à-dire deux entrées ici : la taille en cm (dans ce schéma 'Input 1') et la masse en kg (dans ce schéma 'Input 2') ainsi qu'une troisième entrée constante égale à 1 (ici incarnée par 'Biais').

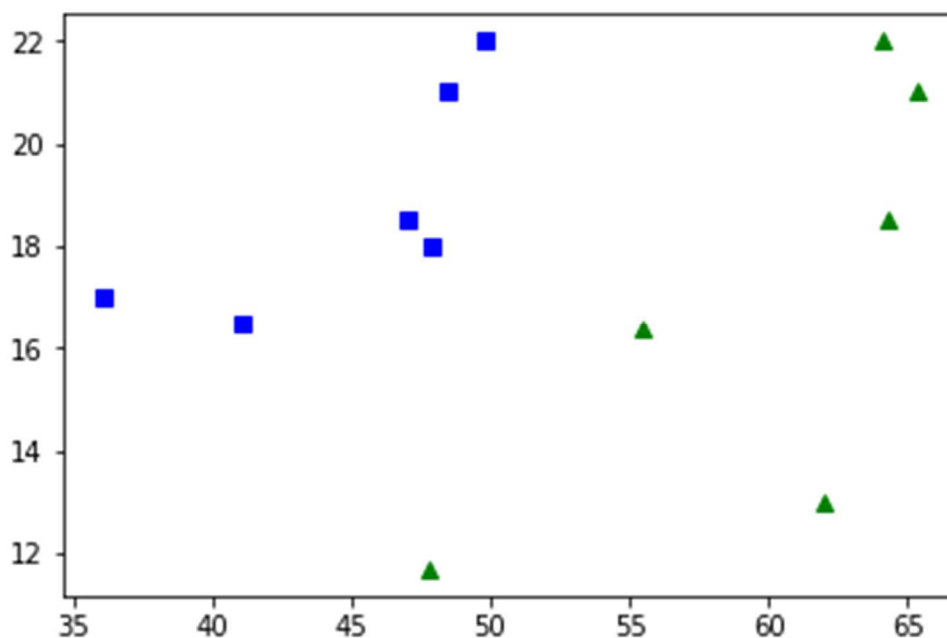
Les poids initiaux sont définis de manière aléatoire et évoluent au fur et à mesure de l'apprentissage. Les notations utilisées sur ce schéma ne correspondent pas aux notations qui suivent.

#### 1.1.2. SCHEMA-BLOC DE L'ALGORITHME

Pour une plus grande clarté du code, nous avons réalisé un schéma-bloc qui permet de voir les différentes actions et interactions entre les fonctions et les gestionnaires d'évènements.



### 1.1.3. REPRESENTATION GRAPHIQUE DES DONNEES D'ENTREES



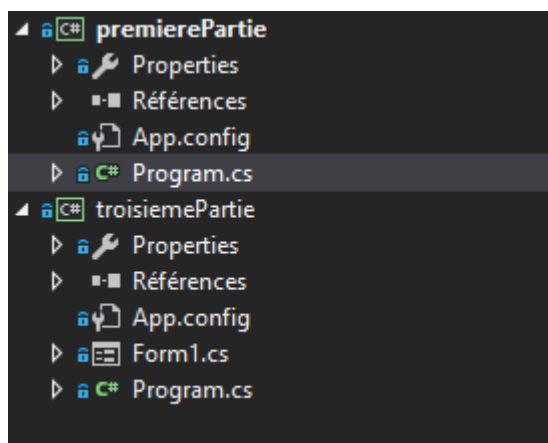
Sur ce graphique, est montré en abscisse la taille des individus et en ordonnées leur masse. Les individus de l'espèce A sont représentés par les triangles bleus et les individus de l'espèce B sont représentés par les triangles vert.

Avec ce graphique, on peut ainsi donner une appréciation de la séparabilité des données et donc des chances de réussite de l'apprentissage. Dans notre cas, il semble que l'algorithme donne une droite séparatrice : les deux groupes étant d'un point de vue macroscopique bien distinct.

---

#### 1.1.4. ARCHITECTURE DU PROJET

La figure ci-dessous est un screen issu de l'exploration de la solution sous Visual Studio 2015. Elle montre les différents projets et documents au sein des projets.



Le programme est un projet Visual Studio Console. Il a donc l'architecture caractéristique de ce format. Le point remarquable est que tout est piloté par un programme *Program.cs*, le rassemblement du code permet une facilité de lecture.

---

## 1.2. REALISATION

---

### 1.2.1. JUSTIFICATION DE CONCEPTION

Notre choix a été de faire un programme sans faire appel au WinForms mais à une console classique, ce qui, pour un exercice de séparation de données, semblait approprié car la validation de l'apprentissage peut être mise en œuvre par une représentation graphique simple et le programme ne nécessite pas spécialement d'affichage graphique contrairement aux autres questions.

---

### 1.2.2. ILLUSTRATION PAR UN CAS D'UTILISATION

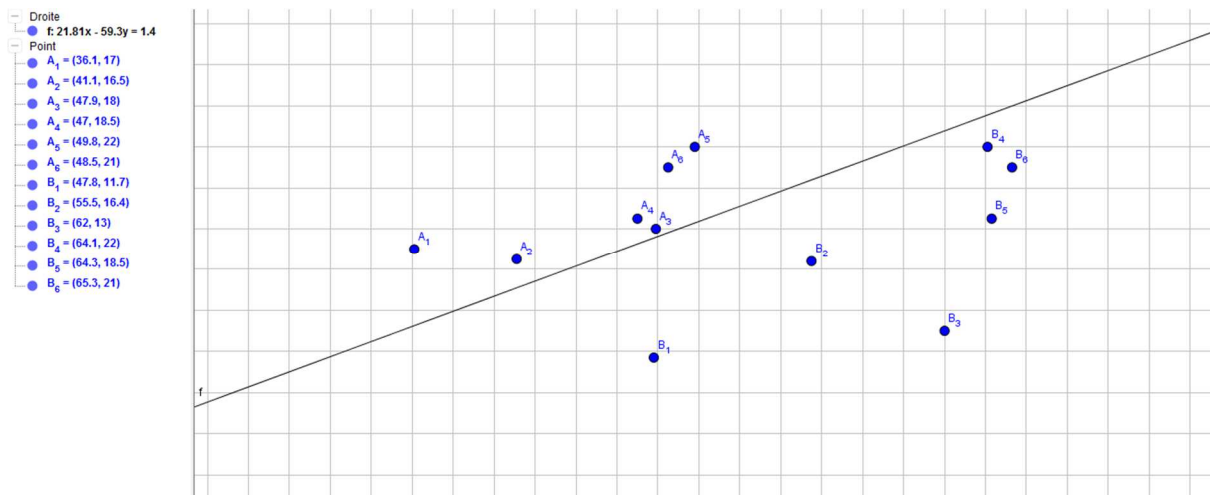
Ici, on a choisi de se placer dans le cas le plus général, à savoir que les poids initiaux des entrées sont choisis de manière aléatoire. De plus, en termes de seuillage, lors du calcul de la sortie du perceptron, on se place dans le cas d'un échelon simple unitaire.

A la compilation, le perceptron apprend par rapport aux données fournies. Un affichage sur la console permet d'obtenir les coefficients de la droite séparatrice. Il suffit après cela, si l'on désire une représentation graphique du résultat de l'apprentissage de tracer sur un graphique les points ainsi que la droite.

Le retour de la console affiche les résultats suivant après apprentissage :

```
Resultats:
W1 = 21,8159445267245
W2 = -59,3090001700488
W3 = -1,43602534869501
Nombre d'iteration : 9
Nombre erreur assignation totale : 16
```

Il ne reste donc plus qu'à vérifier le postulat du perceptron par un tracé de droite qui sera fait à l'aide de GeoGebra, logiciel simple mais efficace et explicite en termes de tracés en deux dimensions.



La droite séparatrice sépare donc bien les données en deux groupes distincts, on peut donc valider l'apprentissage du perceptron.

## 2. PERCEPTRON MULTICOUCHE

### 2.1. PRESENTATION DE LA SOLUTION

#### 2.1.1. DEMARCHE

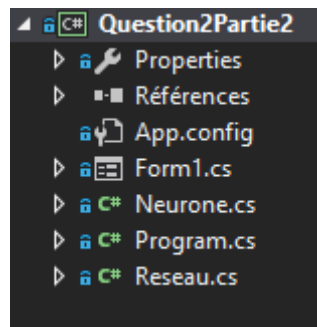
La démarche est la suivante : on crée un perceptron multicouche auquel on apprend à faire une estimation grâce aux entrées disponibles dans *datasetregression.txt*. Ces chiffres permettent au perceptron d'avoir une estimation  $I$  en fonction de deux coordonnées  $x$  et  $y$ .

Suite à cela, on vérifie l'apprentissage en demandant au perceptron de coloriser une image blanche. Pour vérifier l'efficacité du système on affiche également une représentation graphique de l'erreur, c'est-à-dire la différence entre la sortie calculée par le perceptron et la sortie prévue.

---

### 2.1.2. ARCHITECTURE DE LA SOLUTION

Le programme a une architecture similaire à tout projet Visual Studio. Un Windows Form permet la mise en place de l'algorithme et deux classes Neurone.cs et Reseau.cs, fournies en début de projet s'ajoutent.

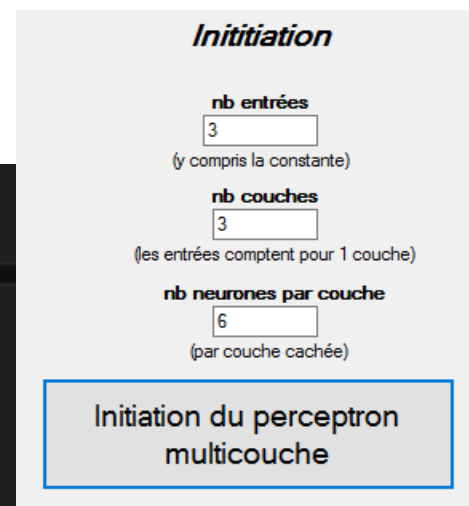


---

### 2.1.3. ILLUSTRATION PAR UN CAS D'UTILISATION

La figure ci-dessous est un screen d'un gestionnaire d'évènement.

```
private void button2_Click(object sender, EventArgs e)
{
    Initialisation des structures graphiques
    #region Apprentissage
    //Apprentissage
    reseau.backprop(dataEntree, dataSortie, 0.8, 10);
    #endregion
    Requete
    Calcul des sorties prevues par le neurone & Conversion en couleur
    Calcul de la difference calculee - prevue & Conversion en couleur
    Affichage des representations graphiques
    pictureBox1.Refresh();
    pictureBox2.Refresh();
}
```



**Initiation**

**nb entrées**  
3  
(y compris la constante)

**nb couches**  
3  
(les entrées comptent pour 1 couche)

**nb neurones par couche**  
6  
(par couche cachée)

**Initiation du perceptron multicouche**

Les différents paramètres pouvant faire varier l'apprentissage sont :

- les paramètres relatifs au réseau lui-même (nombre d'entrées, nombres de couches, nombre de neurones par couche) ;
- les paramètres foncièrement liés à l'apprentissage : soit le coefficient d'apprentissage (intervenant lors de la pondération des poids des entrées) ;
- le nombre d'itération (correspondant au nombre de fois où l'on présente les entrées au réseau).

Choisissons arbitrairement pour cette partie  $\alpha=0.8$  et un nombre d'itération égale à 10.

Perceptron Multicouche

Représentation graphique des sorties calculées

Représentation graphique de l'erreur

**Initiation**

nb entrées  
3  
(y compris la constante)

nb couches  
3  
(les entrées comptent pour 1 couche)

nb neurones par couche  
6  
(par couche cachée)

Initialisation du perceptron multicouche

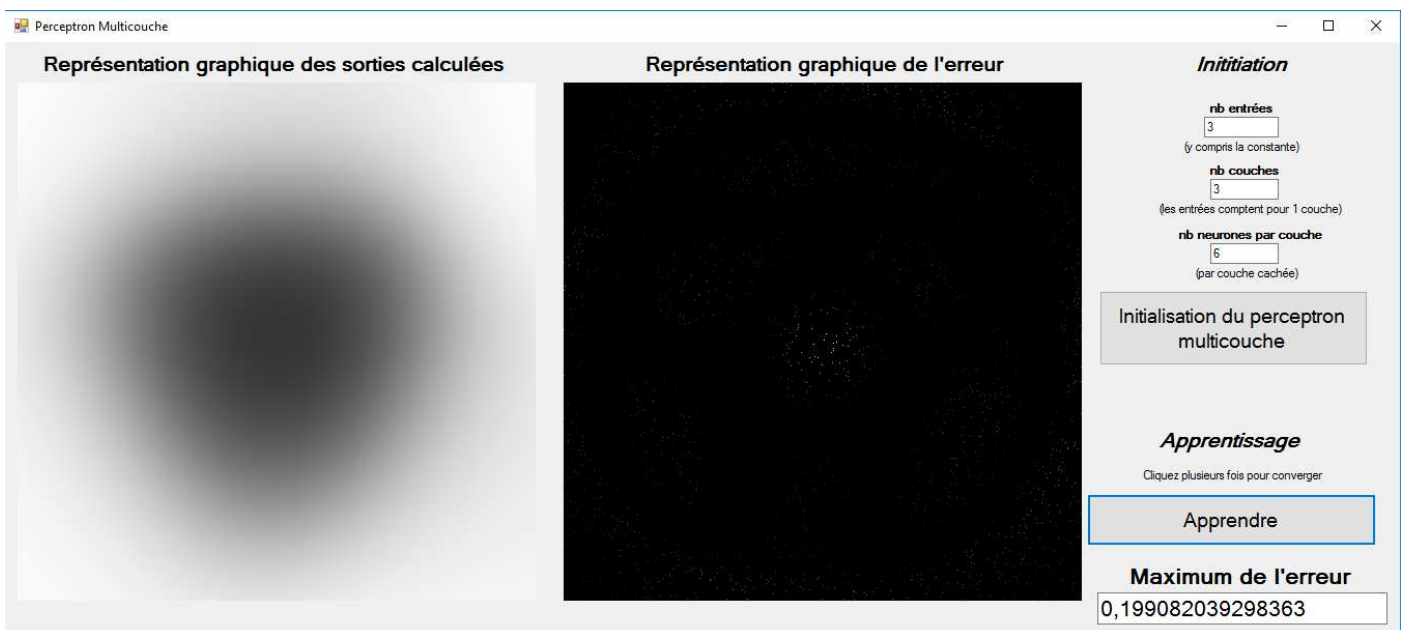
**Apprentissage**

Cliquez plusieurs fois pour converger

Apprendre

Maximum de l'erreur

On initialise le réseau grâce au bouton “Initialisation du perceptron muti-couche”. Puis on clique sur le bouton “Apprendre” pour que le réseau apprenne des data et nous renvoie une représentation graphique de son apprentissage :



On a la représentation de son apprentissage. Le réseau affiche une tache diffuse sur l’image de gauche. Sur l’image de droite on peut observer une représentation de l’erreur avec les points blancs. Dans notre cas, on a une erreur relativement importante en termes de densité d’erreur mais peu important en termes de norme ( $\sim 0.2$  unité d’intensité) dans les pixels centraux affichant un maximum de 0.19 unité d’intensité.

#### 2.1.4. ETUDE D'EFFICACITE DU RESEAU

Le but de cette partie est de trouver un réglage optimal des paramètres pour une complexité temporelle raisonnable (durée de calcul).

##### 2.1.4.1 VARIATIONS DU COEFFICIENT D'APPRENTISSAGE

On évalue en premier lieu l'influence du coefficient Alpha en fixant le réseau (en termes de nombre de neurones) c'est-à-dire un nombre d'entrée égal à 3, 3 couches et 6 neurones par couche. Le coefficient Alpha est le coefficient d'apprentissage, compris entre 0 et 1.

➔ Alpha = 0.2

Représentation graphique des sorties calculées

Représentation graphique de l'erreur

**Initiation**

nb entrées

3

(y compris la constante)

nb couches

3

(les entrées comptent pour 1 couche)

nb neurones par couche

6

(par couche cachée)

Initialisation du perceptron multicouche

**Apprentissage**

Cliquez plusieurs fois pour converger

Apprendre

Maximum de l'erreur

0,708714104425959

➔ Alpha = 0.4

Représentation graphique des sorties calculées

Représentation graphique de l'erreur

**Initiation**

nb entrées

3

(y compris la constante)

nb couches

3

(les entrées comptent pour 1 couche)

nb neurones par couche

6

(par couche cachée)

Initialisation du perceptron multicouche

**Apprentissage**

Cliquez plusieurs fois pour converger

Apprendre

Maximum de l'erreur

0,522515977227263

➔ Alpha = 0.6



Représentation graphique des sorties calculées

Représentation graphique de l'erreur

**Initiation**

nb entrées  
  
(y compris la constante)

nb couches  
  
(les entrées comptent pour 1 couche)

nb neurones par couche  
  
(par couche cachée)

Initialisation du perceptron multicouche

**Apprentissage**  
Cliquez plusieurs fois pour converger

Apprendre

**Maximum de l'erreur**  
0,379334493164845

→ Alpha = 0.8

Représentation graphique des sorties calculées

Représentation graphique de l'erreur

**Initiation**

nb entrées  
  
(y compris la constante)

nb couches  
  
(les entrées comptent pour 1 couche)

nb neurones par couche  
  
(par couche cachée)

Initialisation du perceptron multicouche

**Apprentissage**  
Cliquez plusieurs fois pour converger

Apprendre

**Maximum de l'erreur**  
0,227443247234473

→ Alpha = 1

Représentation graphique des sorties calculées

Représentation graphique de l'erreur

**Initiation**

nb entrées  
  
(y compris la constante)

nb couches  
  
(les entrées comptent pour 1 couche)

nb neurones par couche  
  
(par couche cachée)

Initialisation du perceptron multicouche

**Apprentissage**  
Cliquez plusieurs fois pour converger

Apprendre

**Maximum de l'erreur**  
0,180754847105005

## ➔ Conclusion

On observe au fur et à mesure que le coefficient d'apprentissage augmente, une structure apparaît : une tache centrale. A itération fixe, une pondération plus importante des poids permet une meilleure performance. En termes d'erreur, la différence entre calculé et attendu diminue si le coefficient d'apprentissage augmente. On choisira donc  $\alpha = 0.8$  afin de ne pas se retrouver dans un cas particulier. On voit donc que les erreurs graphiques diminuent, ce qui constitue un point important dans la validation du fonctionnement et de l'apprentissage de notre programme.

### 2.1.4.1. CHOIX DU NOMBRE D'ITERATION

Il est évident qu'il faut augmenter le nombre d'itération. On pourra donc dans le but d'être performant en termes de temps de calcul un nombre d'itérations égal à 1000.

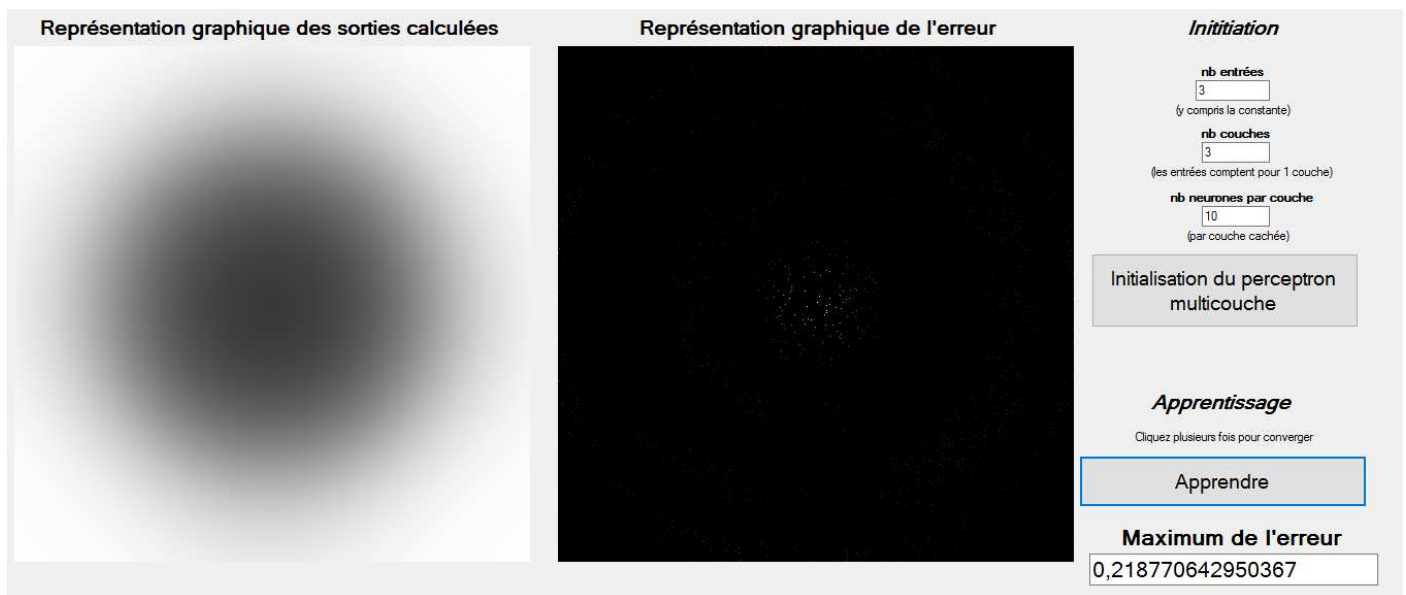
The screenshot displays a web-based interface for a neural network simulation. It is divided into three main sections:

- Représentation graphique des sorties calculées**: A large, empty white rectangular area on the left.
- Représentation graphique de l'erreur**: A large black rectangular area in the center, showing a faint, blurry pattern of light pixels.
- Initiation**: A control panel on the right with the following elements:
  - nb entrées**: A text input field containing the value '3', with the note '(y compris la constante)' below it.
  - nb couches**: A text input field containing the value '3', with the note '(les entrées comptent pour 1 couche)' below it.
  - nb neurones par couche**: A text input field containing the value '6', with the note '(par couche cachée)' below it.
  - Initialisation du perceptron multicouche**: A button.
  - Apprentissage**: A section header with the instruction 'Cliquez plusieurs fois pour converger' and a large 'Apprendre' button.
  - Maximum de l'erreur**: A text input field displaying the value '0,211564787477389'.

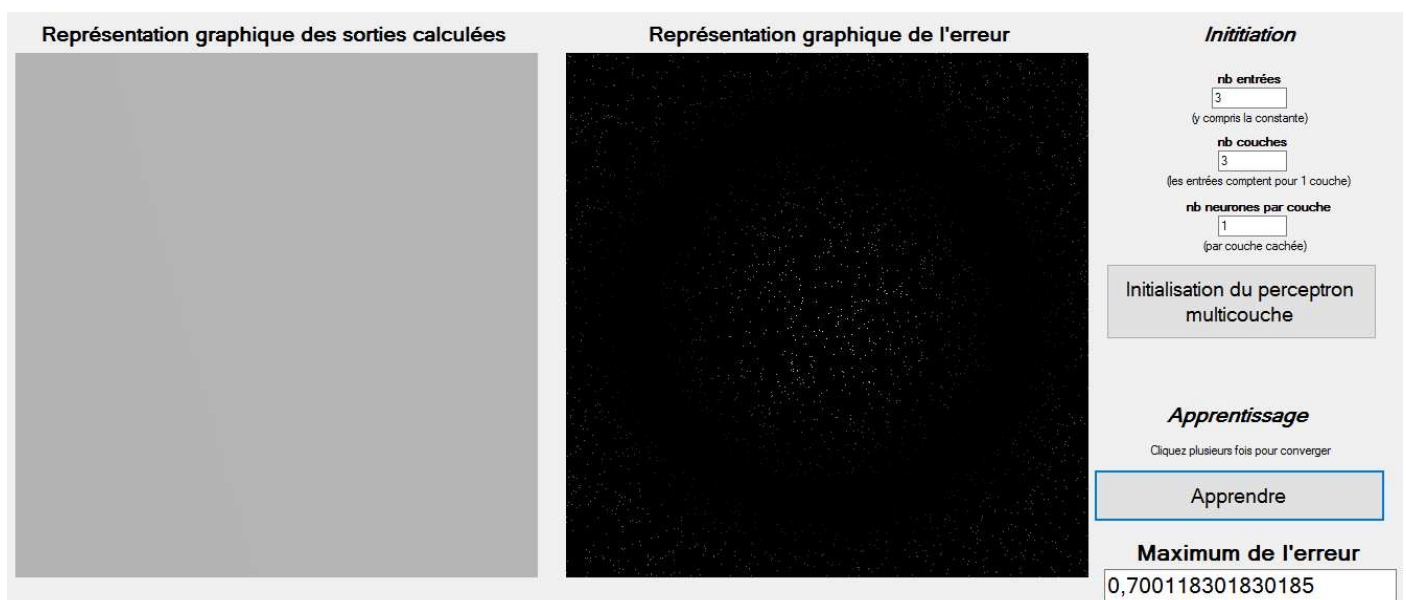
L'erreur diminue et la structure se précise lorsqu'on augmente le nombre d'itération.

Remarque : L'écart entre l'approximation réalisée par un réseau de neurones et la fonction à approcher est inversement proportionnel au nombre de neurones cachés. (Baron, 1993).

Nous allons donc faire deux nouveaux tests pour voir ce phénomène. Dans l'image suivante, on a passé le nombre de neurones par couche à 6 au lieu de 3 dans le cas précédent. On constate donc que l'erreur augmente.



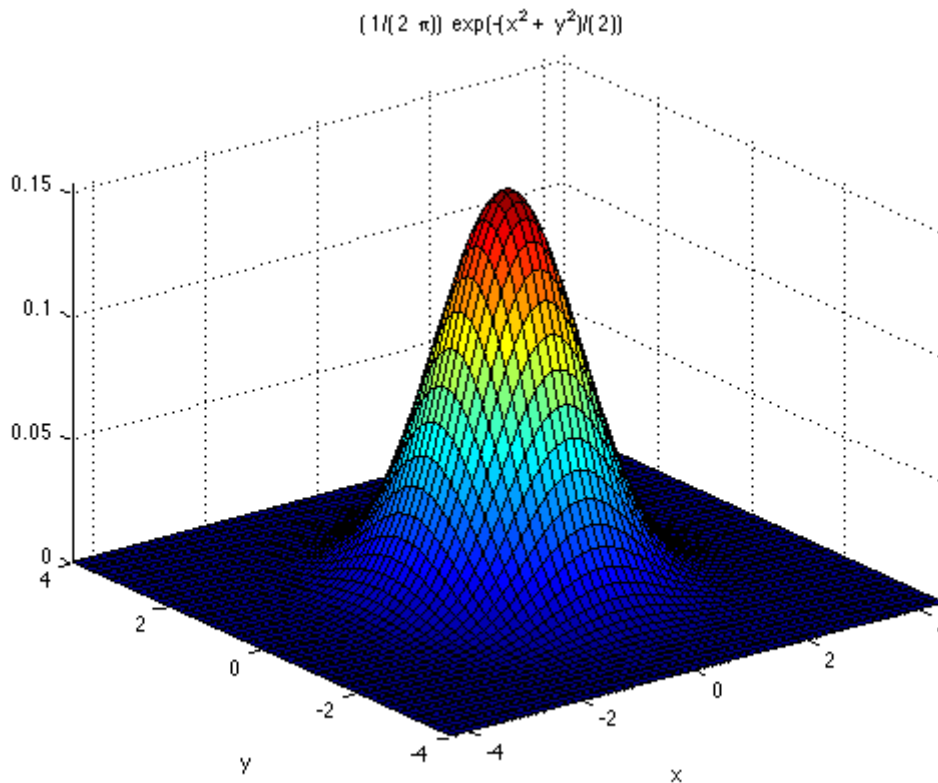
On diminue encore le nombre de neurones par couches pour avoir une comparaison et des résultats plus probants. Cette fois le nombre de neurones est de 1.



Dans notre cas, l'erreur augmente également en si le nombre de neurone des couches cachées diminue.

#### 2.1.4.2. CONCLUSION

L'allure de la fonction que le perceptron tente d'approcher est probablement une gaussienne



### 3. CLASSIFICATION

#### 3.1. APPRENTISSAGE SUPERVISE

Le but de cette partie est de faire apprendre à un programme à différencier deux prototypes de données A et B à l'aide d'un perceptron multicouche. Pour ce faire, on dispose des ressources suivantes :

- Un fichier nommé “dataclassif.txt” contenant une série de 300 couples de valeurs organisés de la façon suivante :
  - Un couple est représenté par trois lignes distinctes
  - La première ligne représente l'index de la valeur dans la liste
  - La seconde ligne représente la première valeur du couple (pour des raisons de praticité, nous la nommerons “x”)
  - La troisième ligne représente la seconde valeur du couple (pour des raisons de praticité, nous la nommerons “y”)
- Une classe “Neurone.cs” qui permet de modéliser un neurone
- Une classe “Reseau.cs” qui permet de modéliser le réseau de neurone utilisé dans l'apprentissage et de définir les différents traitements appliqués sur les données à traiter.

##### 3.1.1. REALISATION

Dans un premier temps, il a été nécessaire de lire le fichier texte et d'entrer ces valeurs dans un tableau d'entrées.

On associe ensuite une sortie attendue à chaque valeur en entrée, pour des raisons pratiques nous avons choisi de prendre la valeur 0,2 pour le prototype A et 0,8 pour le prototype B.

Cette opération est effectuée grâce au code ci-dessous.

```
List<List<double>> lvecteursentrees = new List<List<double>>();
List<List<double>> lve = new List<List<double>>();

// On ouvre le fichier de données en lecture
StreamReader reader = new StreamReader("datasetclassif.txt");

// On a 1 seule sortie associée à chaque vecteur d'entrée
// donc on a seulement 1 liste de réels
// Attention, on suppose ici que le nième élément de cette liste est
// la sortie désirée du nième vecteur de l'ensemble lvecteursentrees
List<double> lsortiesdesirees = new List<double>();
List<double> ls = new List<double>();

// Initialisation des coordonnées du point
double x = 0;
double y = 0;

for (int i = 0; i < 3000; i++)
{
    // On lit les lignes 3 par 3 pour créer les couples de données
    for (int j = 0; j < 3; j++)
    {
        string line = reader.ReadLine();

        if (j == 1)
            x = double.Parse(line);
        else if (j == 2)
            y = double.Parse(line);
    }

    List<double> vect = new List<double>();

    // Ajout des valeurs au vecteur d'entrée
    vect.Add(x/800.0);
    vect.Add(y/800.0);
    lve.Add(vect);

    //Création des sorties désirées
    if (i < 1500)
        ls.Add(0.1);
    else
        ls.Add(0.9);
}
```

De plus, afin d'éviter un maximum le biais lié à la répartition des données dans le fichier (les 1500 premiers couples appartiennent à la classe A et les 1500 derniers couples appartiennent à la classe B), il faut “désorganiser” les données. Pour cela, on utilise un simple mélange aléatoire à l'aide d'une boucle et d'un index choisi aléatoirement.

On utilise la fonction `backprop()` du réseau de neurones afin de présenter un nombre suffisant d'exemples au réseau afin de faire l'apprentissage. Cette fonction va permettre de modifier le poids des neurones du réseau en fonction des entrées et des sorties présentées.

Enfin, afin de vérifier que l'apprentissage ait bien été réalisé, on rentre chaque pixel d'une image 800x800 px et on vérifie si le couple (x,y) désignant le pixel appartient au prototype A (pixel coloré en Bleu) ou au prototype B (pixel coloré en Jaune).

Les résultats de l'apprentissage seront explicités dans la partie suivante.

---

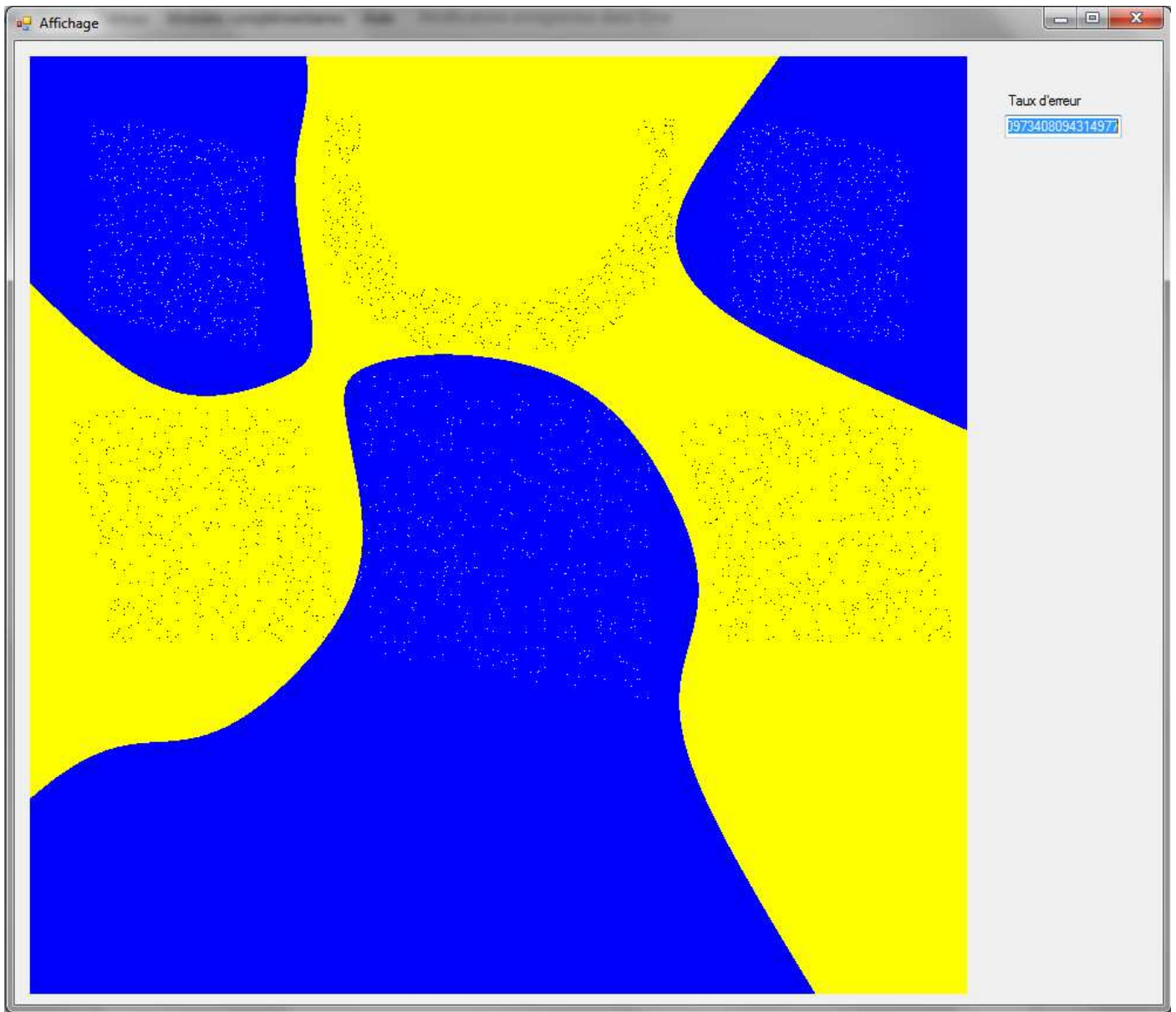
### 3.1.2. RESULTATS

Le tableau suivant donne la distance moyenne entre la sortie attendue et la sortie réelle en fonction des paramètres suivants :

- Nombre de neurones sur la couche cachée ;
- Coefficient d'apprentissage ;
- Nombre d'itérations.

N° de test	Nombre de neurones sur la couche cachée	Coefficient d'apprentissage	Nombre d'itérations	Distance moyenne entre sortie attendue et sortie réelle
1	6	1	1	0,1904
2	8	1	1	0,1860
3	10	1	1	0,1658
4	6	0.1	1	0,2662
5	6	0.5	1	0,1551
6	6	1	100	0,1609
7	6	1	1000	0,1080
8	10	0.5	1000	0,0973

Voici la sortie obtenue pour le test 8 :



Avec les divers tests, on constate que les paramètres de l'apprentissage ont une grande influence sur sa précision. Cependant, il faut faire attention à ne pas surévaluer les paramètres, cela reviendrait à introduire un biais dans le réseau qui rendrait la sortie obtenue peu fiable.



---

### 3.2. APPRENTISSAGE NON-SUPERVISE

Dans cette partie on cherche à réaliser un apprentissage non-supervisé à l'aide de l'algorithme de Kohonen et du fichier *"dataclassif.txt"*.

Ici on a à disposition :

- L'algorithme de Kohonen
- Les données du fichier txt explicitées dans la partie précédente

---

#### 3.2.1. REALISATION

Dans un premier temps, on lit les données du fichier fourni avec la même technique que dans la partie précédente. Ensuite, et ce, afin d'éviter au maximum le biais lié à l'organisation des données, on mélange les entrées.

Nous utilisons ensuite le code fourni pour classer les différents neurones dans les classes. Les données utilisées en entrée par le réseau et provenant du fichier .txt, permettent de modifier le poids des neurones. Nous pouvons rapprocher cela du phénomène d'attraction entre les planètes et les objets stellaires. Imaginons que nous avons des zones avec une forte concentration de petits objets, ils n'appliquent individuellement que peu de force sur une planète (neurone). Mais en grande quantité, ils vont faire dévier petit à petit la trajectoire de la planète. Celle-ci entrant dans le champ de gravité d'autres planètes, elles finissent par se rassembler en un ensemble compact. Cet ensemble compact va former une classe. Reste simplement à calculer pour un point donné, l'influence de chaque neurone pour déterminer sa classe.

La dernière partie était de présenter les résultats obtenus en passant les pixels d'une image 800 x 800px dans le réseau de neurones. Cet affichage est assez simple en termes de code, il suffit de parcourir l'image et de minimiser l'erreur pour chaque neurone de chaque classe, puis d'attribuer la classe du neurone gagnant au pixel utilisé. La principale crainte que nous avons eue à ce niveau était les performances de l'application dans une telle boucle : en effet, nous traitons 640 000 couples de valeurs sur 100 neurones pour le cas le plus simple. Ce chiffre peut très rapidement augmenter si on choisit d'augmenter le nombre de neurones.

Heureusement, cette difficulté ne s'est pas présentée pour un "petit" nombre de neurones mais, passé 400 neurones, l'application commence à mettre beaucoup de temps pour afficher les données.

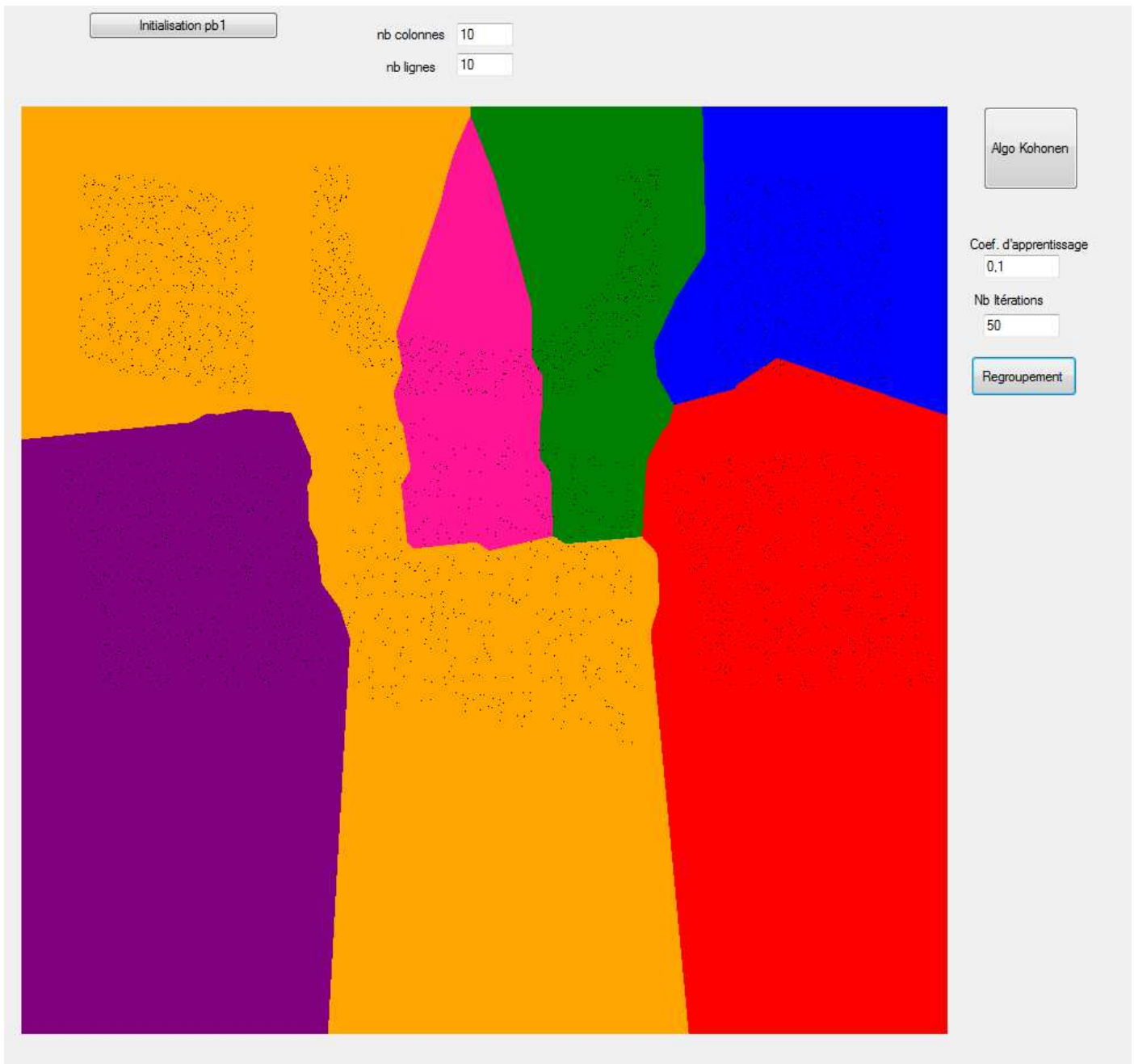
---

#### 3.2.2. RESULTATS

Dans les faits, nous avons eu quelques des résultats étonnants. Nous avons fait varier tous les paramètres de notre réseau, mais il nous a été impossible de récupérer une série de classe adaptée aux entrées que nous avons fournies.

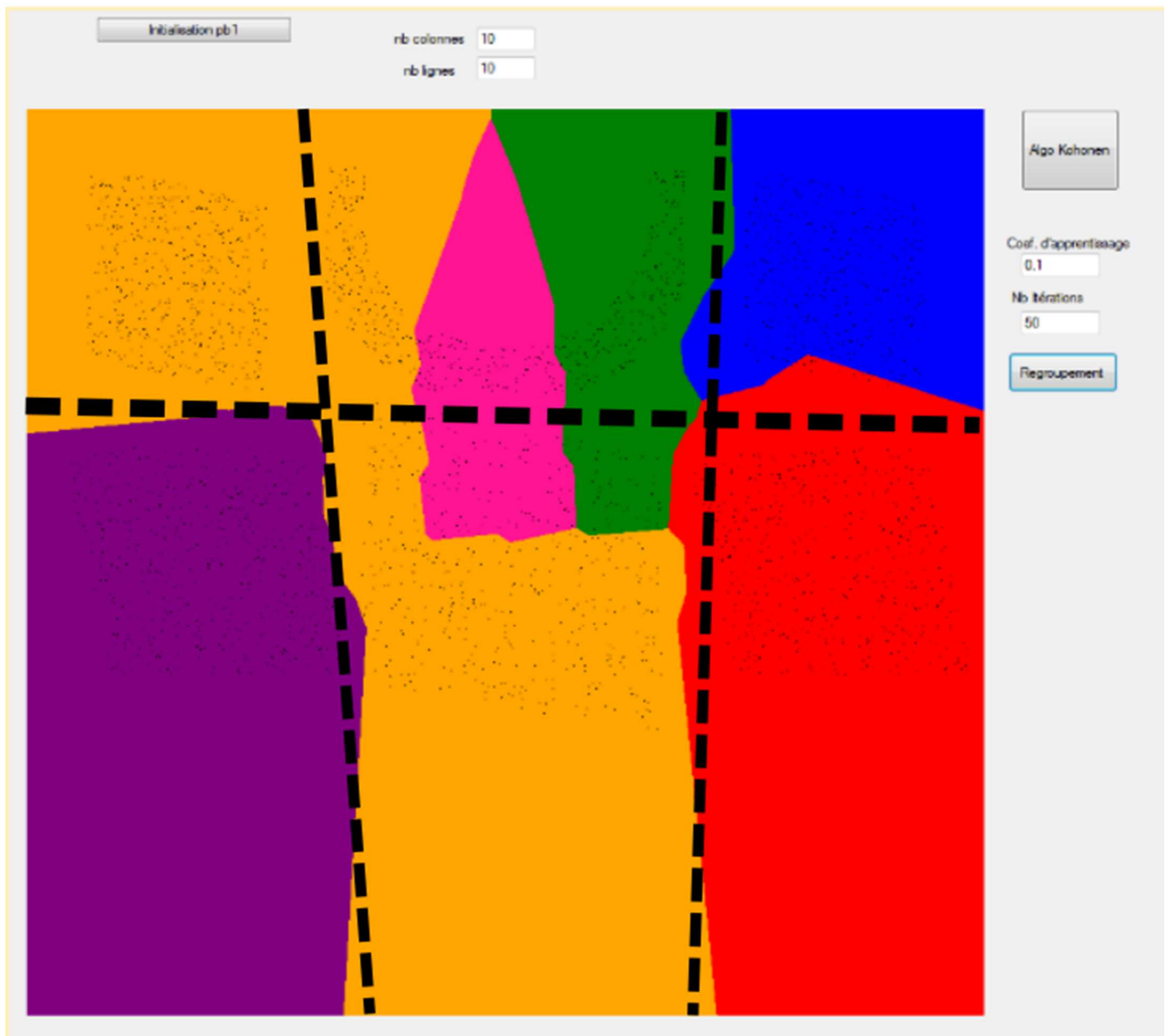
Nous avons des classes formées, comme vous pouvez le voir ci-dessous, mais leur tracé ne correspond pas exactement au tracé théorique attendu.





Ci-dessous, vous pouvez voir le découpage théorique que nous aurions pu espérer obtenir en sortie de l'algorithme de Kohonen. Les 6 classes devraient être nettement découpées et devraient englober les groupes de points obtenus à la lecture du fichier.

Si on compare, on devrait avoir une classe jaune beaucoup plus petite et des classes roses et vertes bien plus grandes de façon à occuper entièrement les cases du haut du schéma. Ce schéma théorique se base sur une séparation nette entre les différents groupes de points, qui correspondent chacun à une même classe.



### 3. GESTION DE PROJET

#### 3.1. REPARTITION DES TACHES

Partie du projet	Question	Marie Guiraute	Grégoire Melin	Alexandre Senaux
Partie 1 :	Question 1	Interface + commentaires des fonctions + Recherche des voisins	Calcul de l'heuristique	Implémentation d'un tableau toute taille + algorithme + calcul du coût de chaque déplacement
	Question 2	Algorithme + Interface 1 deux questions et affichage du détail dans un troisième formulaire	X	Déplacement des chariots sur l'entrepôt affiché via un Thread + place des chariots par le clic
	Question 3	Intégralité	X	X
Partie 2 :	Question 1	X	Intégralité	X
	Question 2	X	Intégralité	X
	Question 3	X	X	Intégralité

#### 3.2. PLANIFICATION

Des séances de travail collectives ont été organisées pour évaluer l'avancement et ensuite se partager correctement les tâches restantes. De cette façon lorsque la seconde partie du projet nous a été donné, nous avons pu nous séparer et travailler sur des documents différents.

La date de rendu des deux projets est le Vendredi 14 Avril à minuit.

## 4. CONCLUSION

---

### 4.1. APPORTS

Nous avons pu faire des progrès sur l’affichage en utilisant des Graphics ou des TreeView. L’utilisation de neurones nous a permis de mieux comprendre les enseignements théoriques du cours.

Les affichages ont permis une grande satisfaction personnelle. Quand les chariots se déplaçaient d’un endroit à un autre, nous pouvions constater et évaluer réellement la performance et l’efficacité de notre programme sur la recherche des plus courts chemins et de l’affichage du déplacement des chariots.

---

### 4.2. POINTS POSITIFS

Pour le déplacement visuel des chariots, nous avons choisi de passer par l’utilisation et la création de bouton qui se colore pour symboliser le passage du chariot. Initialement, nous pensions utiliser un timer pour gérer le déplacement du chariot, mais il nous a paru plus efficace de jouer sur le Thread avec `System.Threading.Thread.Sleep(100);` car nous ne l’avions pas fait en TD.

---

### 4.3. DIFFICULTES RENCONTREES

---

#### 4.3.1. GITHUB ET LE « MERGING »

A trois, nous avons souvent des erreurs de merging au début du projet lorsque nous travaillions tous les trois sur la même partie. Nous avons ensuite séparés les formulaires pour travailler sur des affichages différents puis fait une réunification.

---

#### 4.3.2. CLARTE DU CODE PAR DEFAUT

Nous avons dû prendre beaucoup de temps pour lire et comprendre le code, mais aussi pour savoir comment l’utiliser et quand faire appel aux différentes fonctions déjà codées.