



Rapport de projet

Intelligence Artificielle

Document	
Référence	IAG_RP_001_2017_10_03
Projet	Intelligence Artificielle
Tuteurs	Baptiste PESQUET, Pierre-Alexandre FAVIER, Jean Marc Salotti
Date de début	10 Mars 2017
Membres	
Corentin BIENASSIS	Thibaud LAURENT
	Éric NOUMRI

Sommaire

Sommaire	2
Introduction	3
Partie I (Projet RobotZon)	4
A. Répartition du travail	4
Mode d'emploi	4
B. Question 1 : Explication	6
Création de la classe dérivée NodeWarehouse	6
Constructeur	6
Surcharge des méthodes	6
Choix effectués	6
C. Question 2 : Explication	8
Choix effectués:	9
D. Question 3 : Explication	9
Choix effectués	9
E. Optimisations possibles	10
Partie II	11
A. Répartition du travail	11
B. Question 1 : Classification avec Perceptron 1 couche (Projet NeuralNet)	11
Manuel d'utilisation	11
C. Question 2 : Régression avec Perceptron multi-couches (Projet Regression)	14
Notice de lancement	14
Discussion sur la performance de la classification selon les paramètres	14
D. Question 3 : Classification (Projet Learning)	19
Apprentissage supervisé	19
Influence du nombre de neurones sur la couche cachée	22
Influence du coefficient d'apprentissage (nombre de répétition 500)	23
Influence du nombre de répétition	24
Apprentissage non supervisé	24
Conclusion	33

Introduction

Dans le cadre du module d'Intelligence Artificielle, il nous a été demandé dans un premier temps de créer en C# un programme permettant de modéliser la récupération d'objet via des chariots de livraison au sein d'un entrepôt. Dans un second temps, le projet a consisté à programmer en C# un perceptron permettant de classifier des couples de mesures. Ce rapport détaille l'organisation et les travaux effectués lors de ce projet.

I. Partie I (Projet RobotZon)

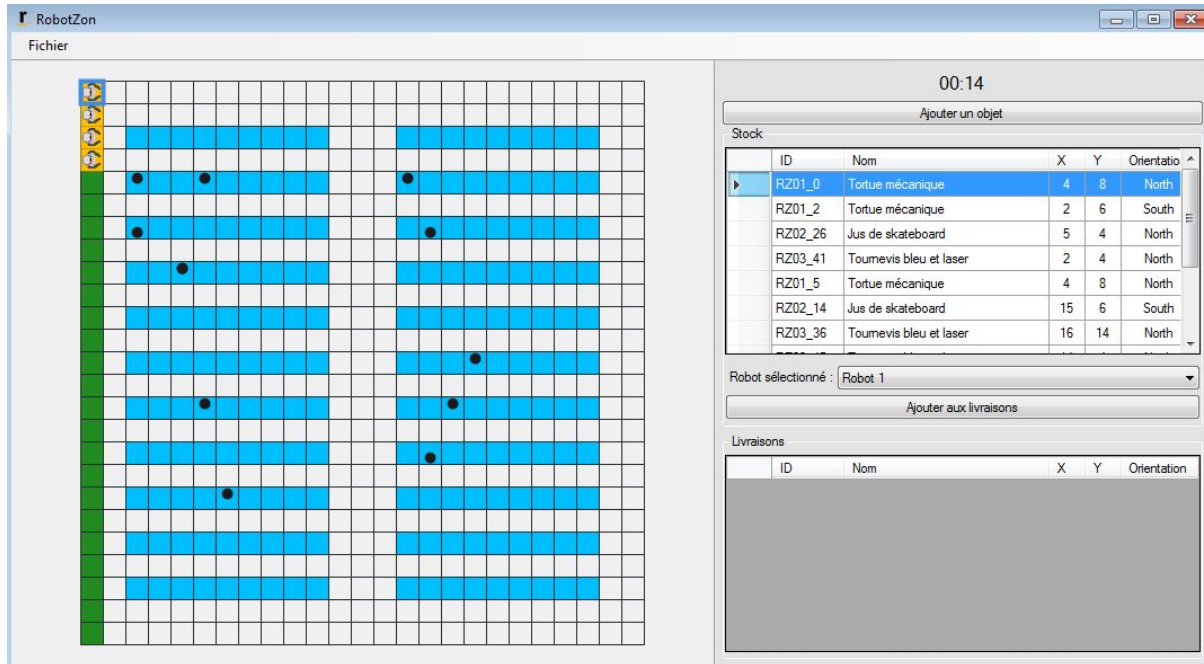
A. Répartition du travail

	Corentin Bienassis	Eric Noumri	Thibaud Laurent
Question 1	Mise en place de l'heuristique	Création du formulaire permettant d'initialiser le nombre de robots	Création de l'interface permettant de visualiser l'entrepôt
Question 2	Mise en place du trajet de retour du robot	Mise en place des contraintes temporelles	Mise en place du changement d'image pendant l'orientation
Question 3	Prise en compte de la trajectoire des autres chariots	Mise en place des tests	Prise en compte de la position de départ des robots

Mode d'emploi

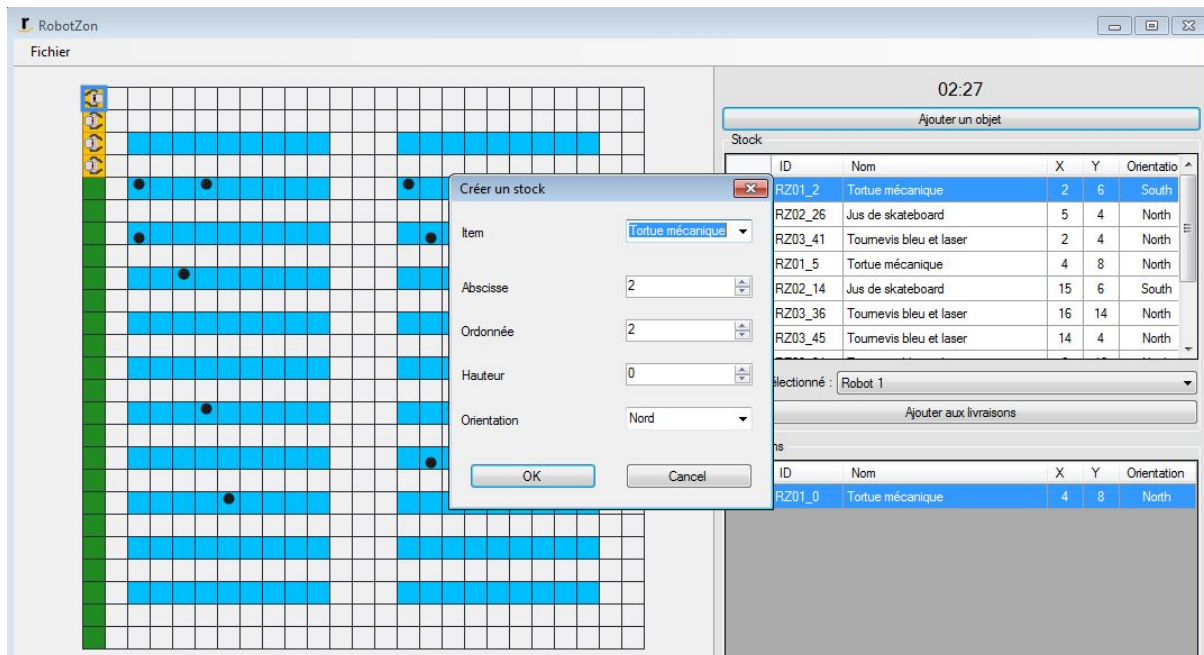
Le premier formulaire qui s'affiche est celui-ci, permettant à l'utilisateur d'initialiser le nombre de robots et leurs positions:

L'utilisateur doit cliquer sur le bouton Attribuer la position pour qu'une position qu'il a déterminé soit associé à un robot sélectionné. Une fois les positions définies, il doit appuyer sur le bouton valider.



Ce formulaire permet à l'utilisateur de demander à un robot de chercher un objet. Pour sélectionner un robot, l'utilisateur peut cliquer dessus. Il peut ensuite envoyer le robot chercher un objet en sélectionnant un objet sur le tableau situé en haut à droite, et en appuyant sur le bouton Ajouter aux livraisons.

L'utilisateur peut également ajouter un nouvel objet au sein de l'entrepôt en appuyant sur le bouton ajouter un objet. Un formulaire s'affiche alors, lui demandant de définir la position du nouvel objet.



B. Question 1 : Explication

1. Création de la classe dérivée NodeWarehouse

Constructeur

```
public NodeWarehouse(Warehouse warehouse, Robot robot, Position position, Position objective) : base()
```

Avec la signature de notre constructeur, on peut noter qu'il prend certains arguments en paramètre.

- Les arguments *position* et *objective* sont utiles pour les méthodes surchargées de la classe, par exemple, la méthode *isEqual()*.
- Les autres arguments, à savoir *warehouse* et *robot* permettent à la méthode *getListSucc()* de récupérer les successeurs en fonction de l'environnement courant de l'entrepôt.

Cela signifie que chaque *NodeWarehouse* possèdent chacun ces attributs. Cette organisation peut paraître peu optimisée, mais comme il ne s'agit que de références, les objets en mémoire ne sont pas stockés pour chaque noeud; seul leur adresse est stockée.

Surcharge des méthodes

Pour ne pas surcharger le rapport, regardons de plus près uniquement les deux méthodes surchargées les plus intéressantes.

getListSucc: il s'agit sans doute de la méthode la plus importante. Elle permet à l'algorithme A* de rechercher les noeuds suivants du graphe. Dans notre cas, on ajoute à la liste des successeurs

1. la position actuelle
2. les positions adjacentes qui ne sont ni des étagères ni des robots ni qui ne contiendront un robot en même temps que

Sachant que la liste des successeurs ne doit pas contenir les robots qui seront à une position X à un temps T futur, ce que nous prenons en compte.

getHCost : Cette fonction est facultative dans le code mais n'en reste pas moins intéressante. Afin de réduire le temps de calcul de l'algorithme A*, il est indiqué à ce dernier que le poids des cases les plus éloignées (à vol d'oiseau) de notre objectif est plus important. Ainsi, l'algorithme favorisera dans sa recherche les successeurs les plus proches de l'objectif.

Choix effectués

Avant de pouvoir visualiser l'entrepôt, l'utilisateur accède tout d'abord à un formulaire permettant de définir le nombre de robot. Cela permet à ce dernier de prendre conscience de ses ressources avant de lancer le programme, et de définir ces dernières avec soin. La position d'un robot n'est pas attribué tant que l'utilisateur ne clique pas sur le bouton attribuer la position. Il nous a paru important de demander à l'utilisateur de valider une position assigné avant de lancer le programme.

Nous avons décidé de représenter l'entrepôt selon un tableau, où certaines cases sont mises en valeurs selon un code couleur particulier:

- Les cases en vertes représentent la zone de départ des chariots. Une fois après avoir récupéré un objet, les chariots reviennent dans cette zone de départ.

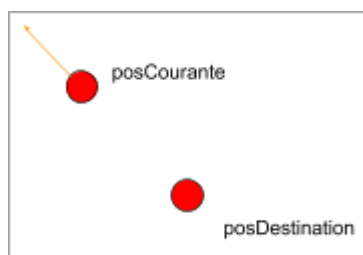
- Les cases constituant la trajectoire d'un chariot sont colorés en oranges.
- Les cases représentant les étagères sont colorés en bleus. Si l'étagère contient un objet, un rond noir est dessiné au sein de la case.

La sélection d'un robot se fait notamment en cliquant dessus. Cela nous a paru comme étant une action rapide et instinctive pour sélectionner un chariot.

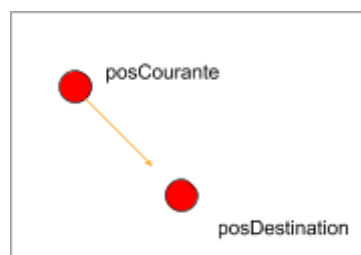
Les objets sont représentés au sein d'un tableau, faisant office d'inventaire. Il nous a semblé nécessaire d'indiquer les différents objets disponibles ainsi que leurs coordonnées dans le même formulaire permettant d'assigner un objet à aller cherché à un robot.

L'heuristique utilisée est la suivante : lorsque l'on récupère la liste des successeurs, on ajoute une valeur au noeud courant, qui correspond à la distance à vol d'oiseau de la position courante jusqu'à l'objectif. Plus le noeud est loin de l'objectif, plus cette valeur (HCost) sera élevée.

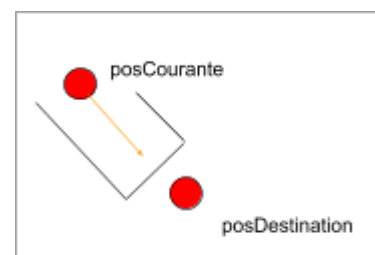
Lorsque l'algorithme A* s'exécute, il va prioriser dans son parcours de graphe les noeuds avec le poids le plus faible. Ainsi, il trouvera le meilleur chemin en moins de temps (ou, au pire, en un temps égal) que s'il ne connaissait pas cette valeur HCost.



Cas sans heuristique



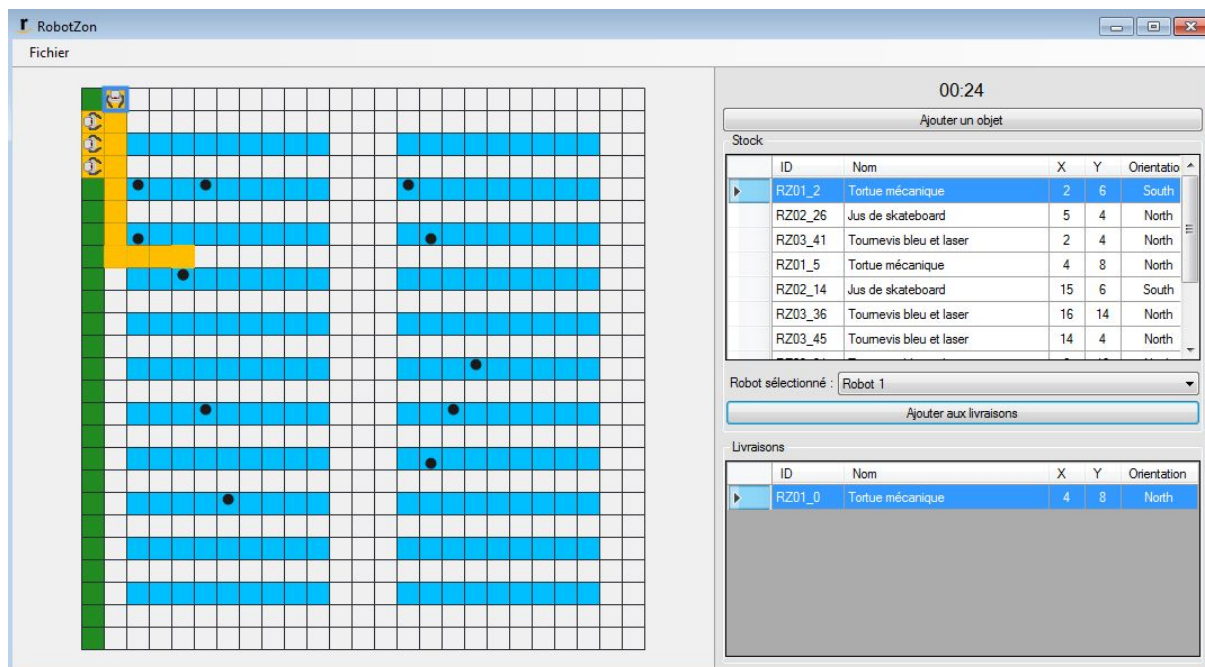
Cas avec heuristique



Pire cas avec heuristique

La flèche orange indique la direction priorisée lors du choix des successeur par l'algorithme A* avec et sans heuristique

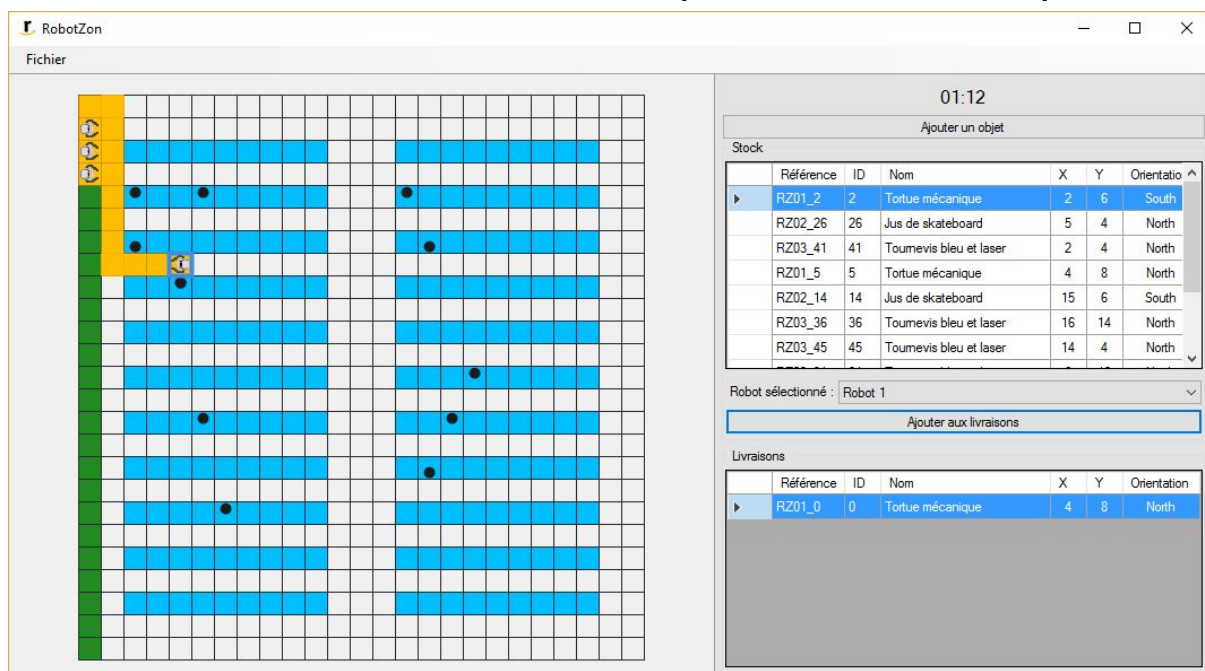
Test fonctionnel: aller chercher un objet



On sélectionne un chariot, puis on lui demande d'aller chercher un objet situé en (2;6).

C. Question 2 : Explication

Test fonctionnel: aller et retour et mise en place des contraintes temporels



Un chariot est utilisé afin de prendre un objet en (4;8). On vérifie que le robot prend bien une seconde pour se déplacer en ligne droite, et 3 secondes pour tourner, ainsi que 10 secondes pour

recupérer un objet en restant immobile face à l'objet qu'il doit récupérer. Une fois l'objet récupéré, le chariot on s'assure que le chariot revient bien à sa position de départ.

Choix effectués:

Afin de respecter les contraintes temporelles, nous avons décidé que pour qu'un robot se déplace d'une case, il mettrait toujours une seconde. Afin de pouvoir simuler le temps d'attente, pour récupérer un objet ou pour tourner, de rajouter plusieurs la même case au sein de la liste de déplacement du chariot.

D. Question 3 : Explication

Test fonctionnel: prise en compte de la trajectoire des autres chariots

The screenshot shows the RobotZon application window. On the left is a grid-based simulation environment with a yellow starting area on the left edge. Several black dots representing robots are positioned on the grid. On the right is a control panel with a timer showing 00:49. It includes a 'Stock' table with a list of objects, a 'Robot sélectionné' dropdown set to 'Robot 2', and a 'Livraisons' table showing the current delivery task.

Référence	ID	Nom	X	Y	Orientatio
RZ01_0	0	Tortue mécanique	4	8	North
RZ02_26	26	Jus de skateboard	5	4	North
RZ03_41	41	Toumevis bleu et laser	2	4	North
RZ01_5	5	Tortue mécanique	4	8	North
RZ02_14	14	Jus de skateboard	15	6	South
RZ03_36	36	Toumevis bleu et laser	16	14	North
RZ03_45	45	Toumevis bleu et laser	14	4	North

Référence	ID	Nom	X	Y	Orientatio
RZ01_5	5	Tortue mécanique	4	8	North

Un chariot est envoyé chercher un objet en (4;8). On envoie ensuite un chariot chercher un objet en (2;6). Lors du retour, le premier chariot envoyé attend que le chariot récupère son objet avant de revenir à sa position de départ.

Choix effectués

Afin de prendre en compte le déplacement des autres chariots, nous avons procédé de la manière suivante: lorsque l'utilisateur assigne un objet à aller chercher à un chariot, ce dernier détermine si une case sera occupé par un chariot, en vérifiant si le déplacement qu'il prévoit comprend une case dans laquelle un chariot sera au même moment.

E. Optimisations possibles

En raison d'un manque de temps, certaines fonctionnalités envisagées n'ont pas pu être implémentées :

- Il aurait été plus ergonomique que l'utilisateur puisse cliquer sur un objet situé dans l'entrepôt, ce qui aurait automatiquement engendré une récupération de cet objet par le robot le plus proche.
- Une fois après avoir récupéré un objet, le robot devrait pouvoir revenir à la zone de départ la plus proche. Dans notre programme, le robot, une fois après avoir récupéré un objet, se dirige vers la zone de départ non occupé la plus proche de la case (0;0).

II. Partie II

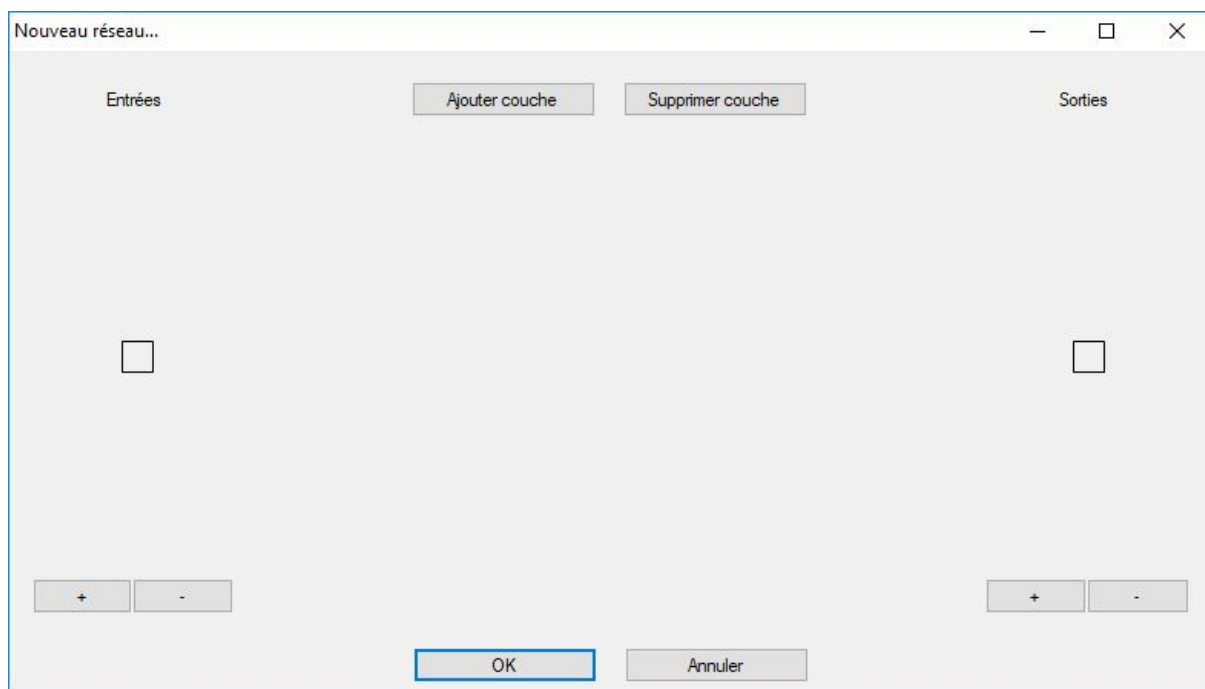
A. Répartition du travail

	Membre du groupe
Question 1	Thibaud Laurent
Question 2	Corentin Bienassis
Question 3	Eric Noumri

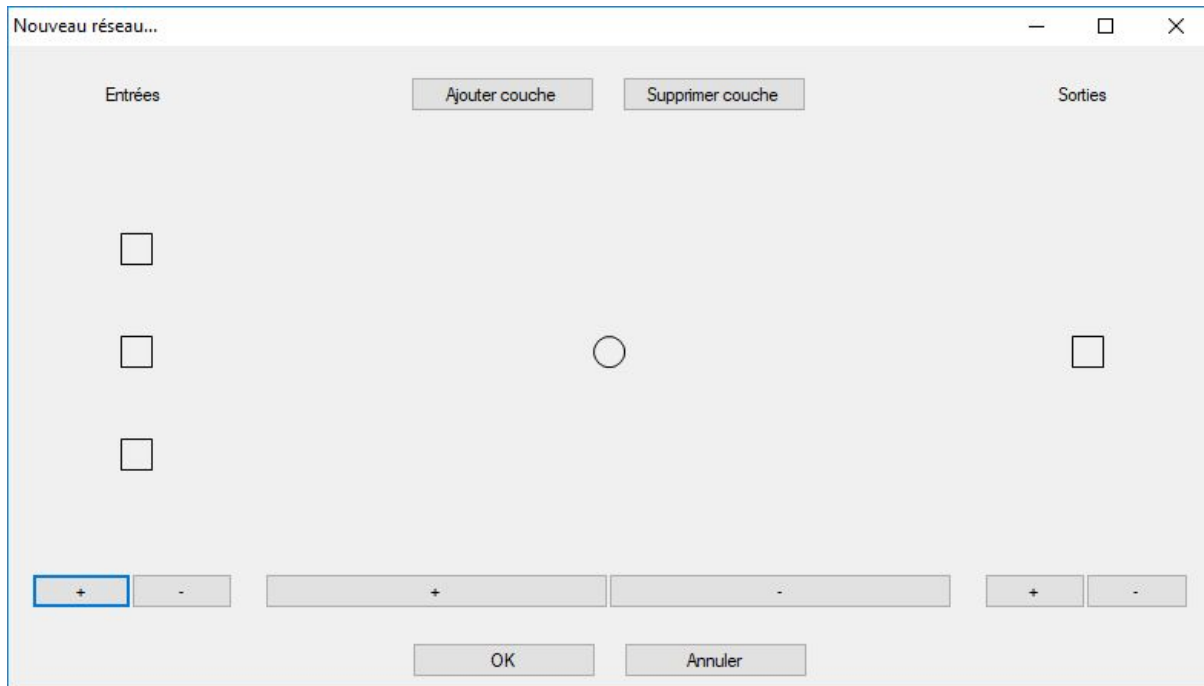
B. Question 1 : Classification avec Perceptron 1 couche (Projet NeuralNet)

Manuel d'utilisation

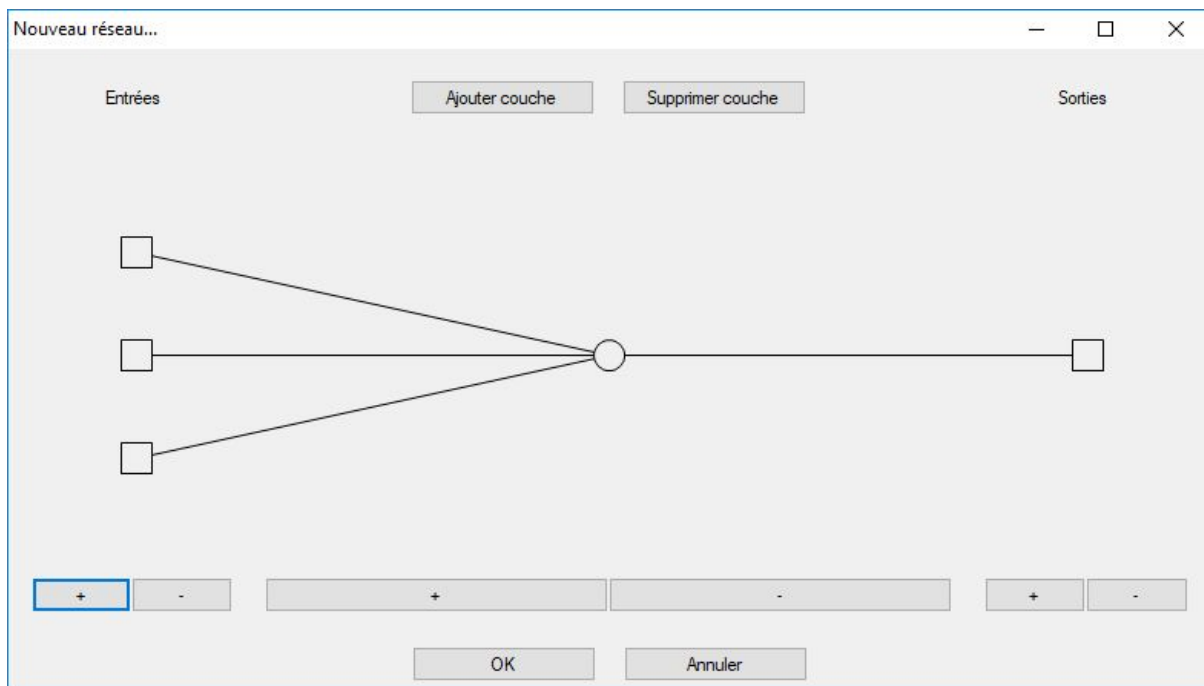
Nous allons détailler comment mettre en place un perceptron. Nous avons créé un éditeur de perceptrons, accessible dans le menu "Fichier", puis "Nouveau Perceptron". Une fois dans ce créateur, il suffit de cliquer sur "Ajouter une couche" et "Supprimer une couche" pour contrôler le nombre de couches du perceptron (10 couches maximum) ; puis sur les boutons "+" et "-" de chaque couche pour ajouter ou supprimer un neurone (10 neurones maximum).



Ainsi, créons un perceptron identique à celui de l'exercice :

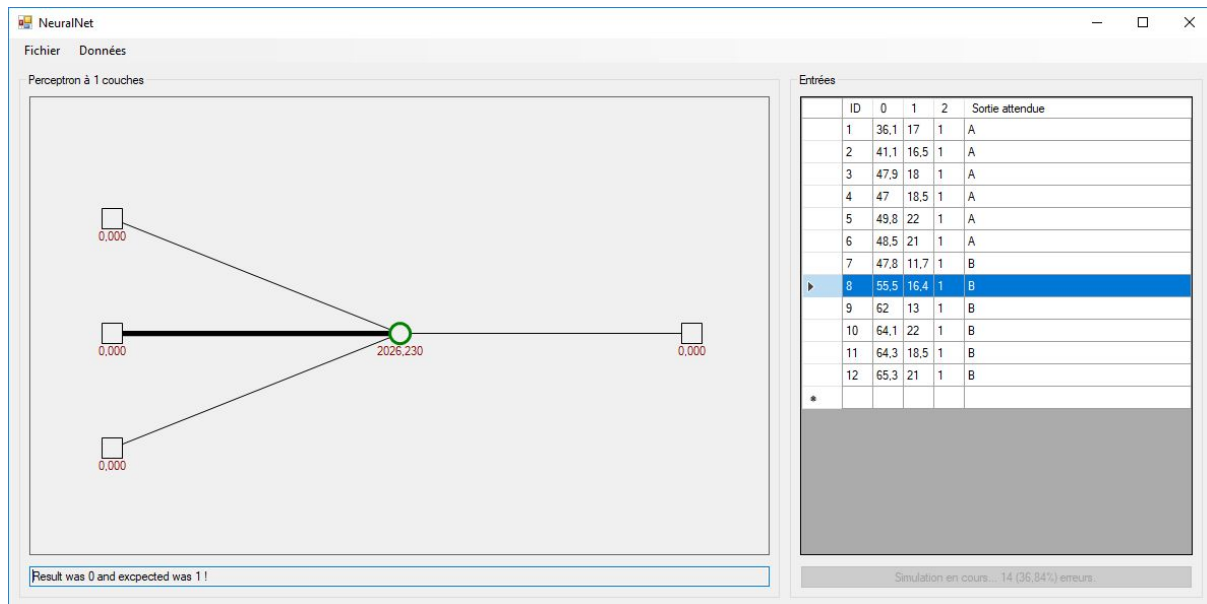


Pour lier des neurones par des poids, il suffit de cliquer sur un neurone (une liaison temporaire sera créée) et de cliquer sur un second neurone. Notons qu'il n'est possible de créer une liaison que entre un neurone d'une couche, et un neurone de la couche suivante.



Puis appuyons sur "OK". Ensuite, il faut charger les données qui permettront au perceptron d'évoluer. Pour cela, il suffit d'aller au menu "Données" puis "Charger" et d'ouvrir un fichier valide grâce à la boîte de dialogue qui apparaît. Enfin, appuyons sur OK.

Enfin, il ne nous reste qu'à sélectionner une entrée de départ, puis appuyer sur "Lancer la simulation". Toutes les 10 millisecondes, une nouvelle entrée aléatoire sera donnée au perceptron, et les poids des liaisons entre la couche d'entrée et le neurone central seront modifiés en conséquence.

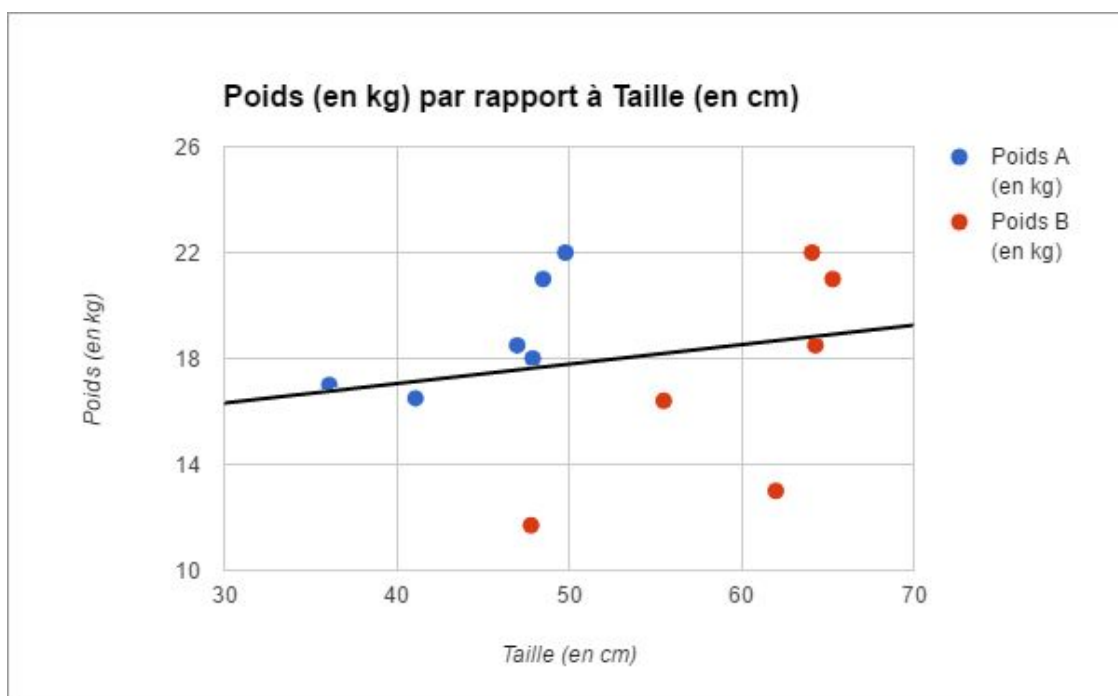


Avec un seuil arbitraire fixé à 1000, le perceptron se bloque vers 147 erreurs et n'en fait plus aucune avec les poids suivants :

$$W1 = -117,2$$

$$W2 = 373,3$$

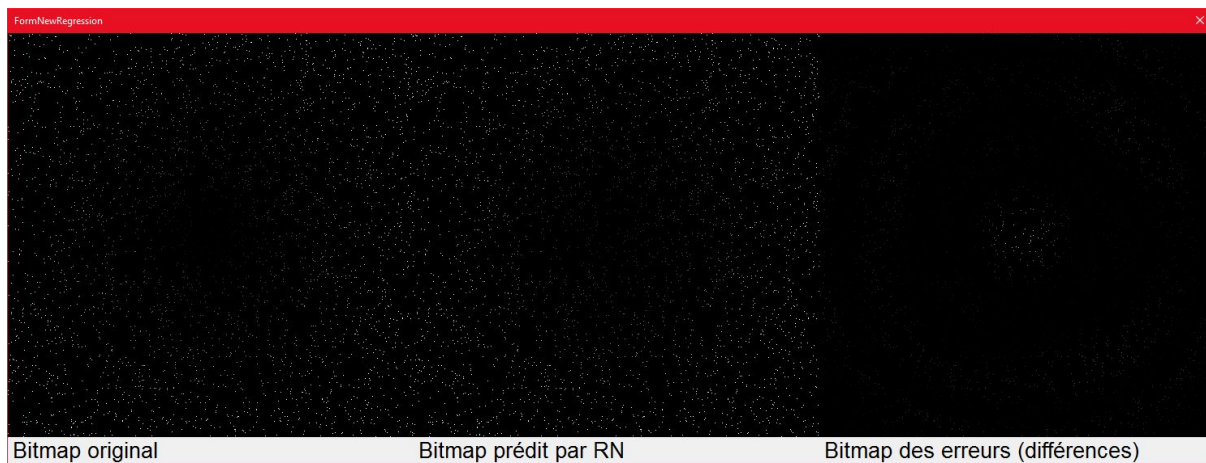
$$W3 = 22$$



C. Question 2 : Régression avec Perceptron multi-couches (Projet Regression)

Notice de lancement

1. Ouvrir le programme *Genetik* avec Visual Studio
2. Désigner *Regression* comme projet de démarrage
3. Démarrer l'exécution. La fenêtre suivante se lance.



Discussion sur la performance de la classification selon les paramètres

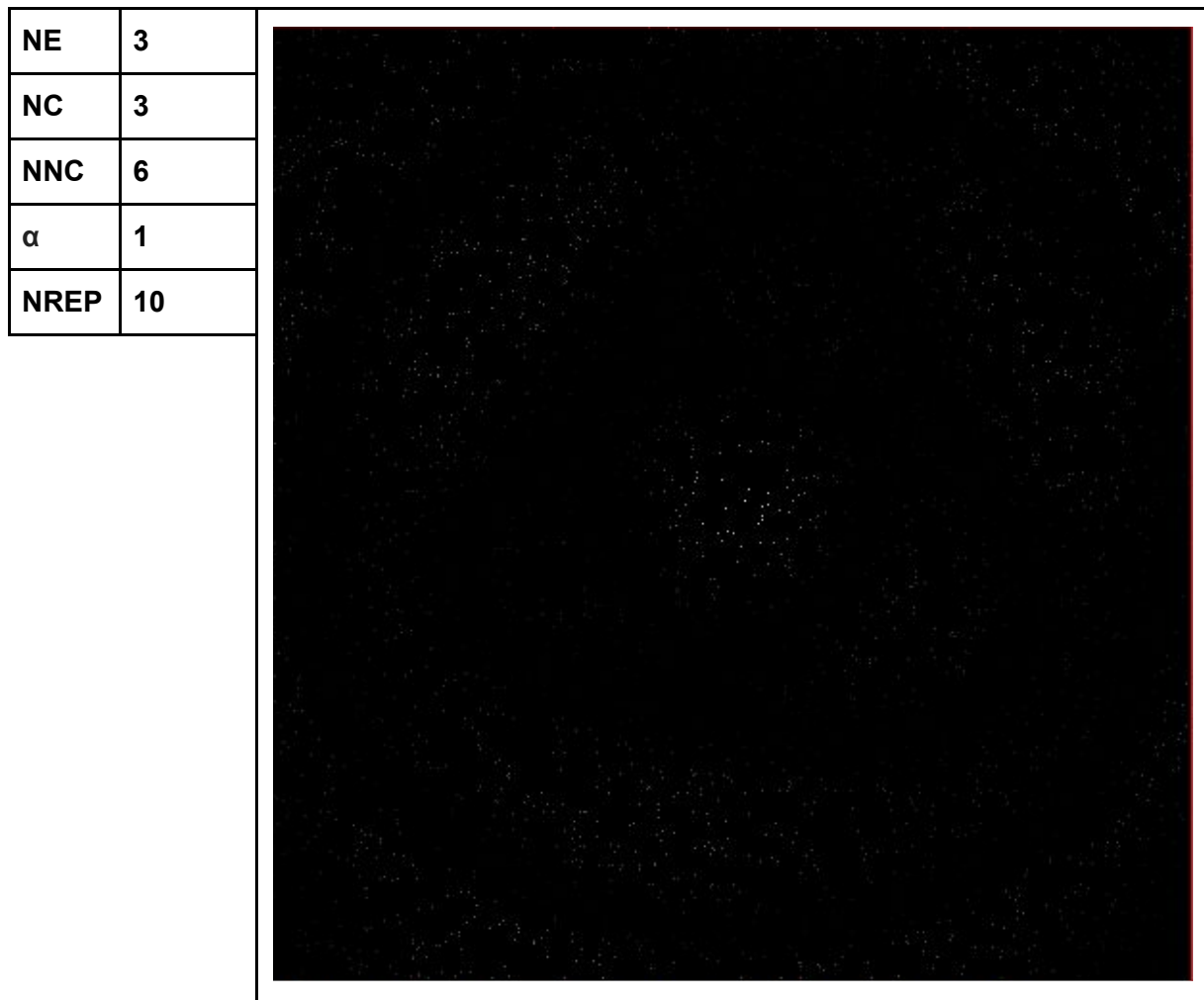
Comme on peut s'en douter, la qualité de la classification dépend des paramètres du réseau neuronal ainsi que de ceux de la fonction de rétropropagation. Nous avons fait quelques tests en changeant ces paramètres. Les résultats sont exposés dans la partie ci-dessous.

Comme le montre la figure des 3 bitmaps au dessus, le bitmap de gauche affiche l'image originale, dont les intensités sont directement issues du fichier *datasetregression.txt*.

Au milieu se trouve le bitmap dont les intensités sont issues du réseau neuronal.

À droite se trouve le bitmap représentant les erreurs entre les sorties du réseau neuronal et les sorties attendues. Concrètement, plus l'écart entre deux pixels respectifs est important, plus l'intensité de ce pixel dans le bitmap de droite sera forte (la valeur de gris est pondérée en fonction de l'erreur maximale).

Pour exposer les différences de performance de classification selon les paramètres, nous afficherons l'image la plus pertinente : celle représentant les erreurs.



Avec,

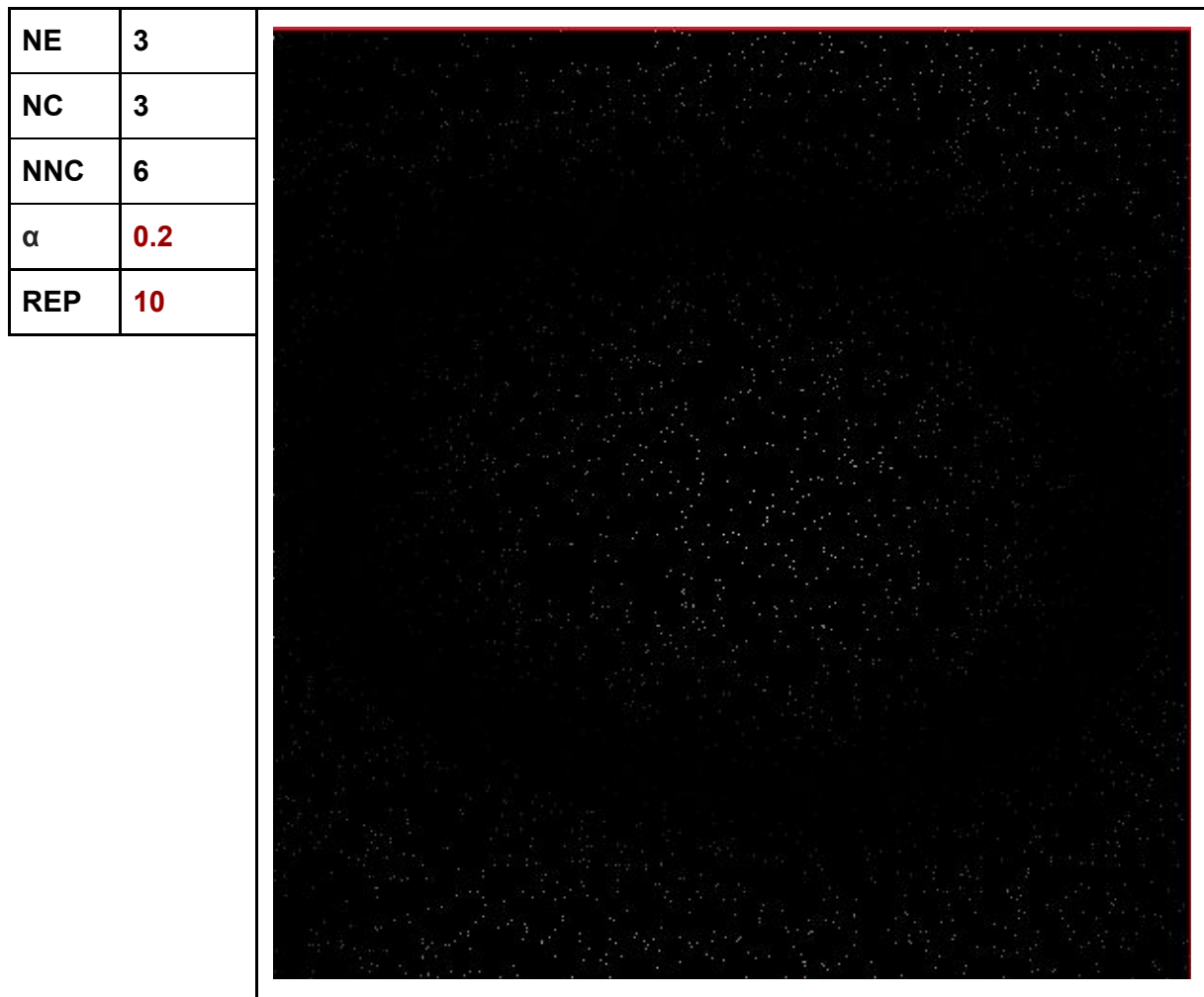
NE : le Nombre de Neurones en Entrée du perceptron

NC : le Nombre de Couches (incluant les couches d'entrée et de sortie)

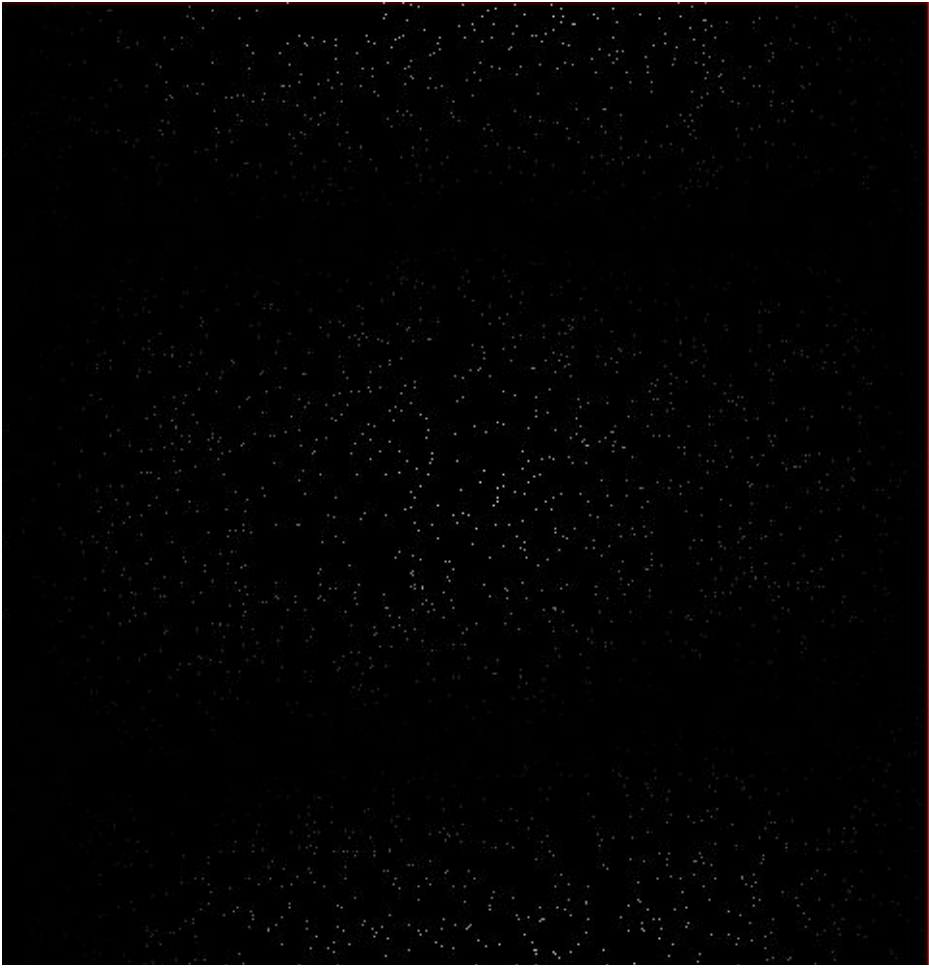
α : le coefficient d'apprentissage lors de l'appel à *backprop()*

NREP : le nombre de répétitions lors de l'appel à *backprop()*

Comme on peut le voir avec cette forme ondulatoire, le réseau neuronal n'a pas réussi à reproduire à 100% la fonction originelle. Les points au centre ayant le plus d'intensité, on en déduit que le réseau neuronal a des difficultés à estimer les valeurs lorsque les coordonnées x et y sont très proches.

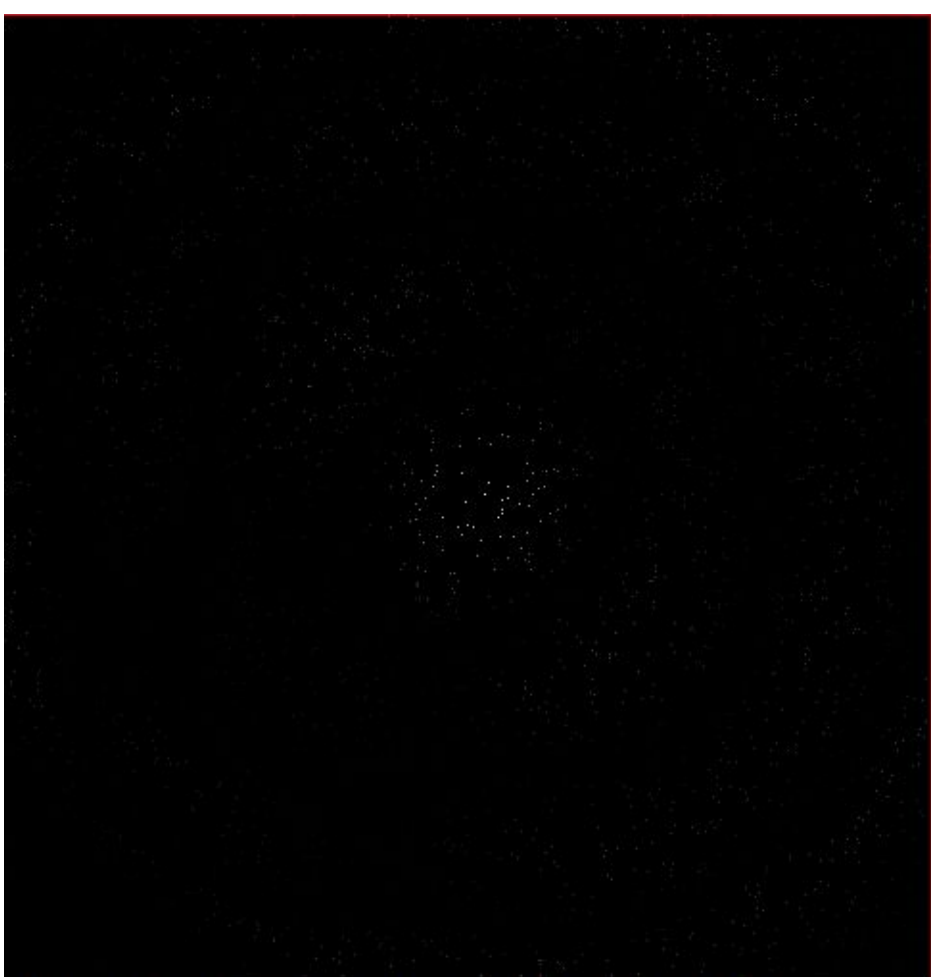


Ici, la présence de nombreux pixels blancs indiquent une erreur plus importante que sur l'exemple précédent. Cela s'explique par le faible coefficient d'apprentissage et par un nombre d'itérations trop faible. Les poids des neurones sont modifiés faiblement à chaque itération et n'ont pas le temps d'atteindre leur poids optimal. Si l'on augmentait le nombre de répétition à 100, on observerait une erreur amoindrie (théorie vérifiée lors de l'écriture de ce rapport).

NE	3	
NC	3	
NNC	2	
α	1	
REP	1000	

Ici, on réduit drastiquement le Nombre de Neurones par Couche (NNC) à 2, au lieu des 6 précédents. On remarque que là encore, les erreurs sont importantes. En effet, le réseau neuronal n'est pas assez complexe pour représenter la fonction originelle. Ici, peu importe le nombre d'itérations ou de répétitions, le résultat ne sera jamais significativement meilleur que celui là.

NE	3
NC	3
NNC	6
α	0.5
NREP	1000



Ici, on obtient un résultat très satisfaisant également. On en déduit qu'il n'existe pas qu'un seul et unique quintuplet de paramètres (NE, NC, NNC, α , REP) permettant d'obtenir un réseau neuronal imitant de manière satisfaisante la fonction originelle.

Bien sûr, si le réseau neuronal n'est pas assez complexe (NE trop faible, NC trop faible ou NNC trop faible), il sera impossible d'avoir un résultat satisfaisant. Par contre, Il semble exister de très nombreux couple (α , REP) pour lesquels le réseau est performant.

Il faut donc trouver le meilleur compromis entre le nombre de répétition (intrinsèque à la puissance de calcul et au temps d'exécution) et le coefficient d'apprentissage (plus il est petit, plus il trouvera une solution optimale - si le nombre de répétitions le permet).

Question subsidiaire : quelle est la fonction mathématique approximativement encodée par le perceptron ?

Le fonction encodée approximativement par le perceptron est une fonction qui corrèle l'intensité z aux coordonnées x et y .

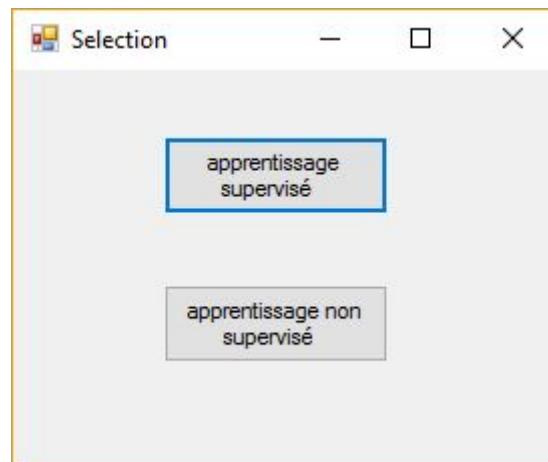
Une fonction qui pourrait correspondre serait : $z = \sqrt{x^2 + y^2}$

Autrement dit, plus la distance entre x et y est faible, plus l'intensité est faible.

Si ce n'est pas forcément cette fonction précise, il s'agit en tout cas d'une fonction radiale, c'est à dire d'une fonction qui affiche une valeur pour un point en fonction de sa distance à l'origine.

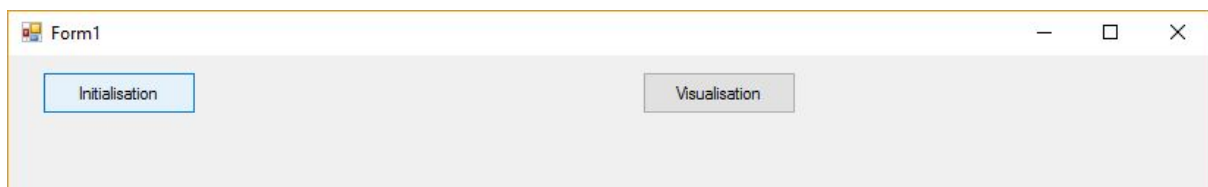
D. Question 3 : Classification (Projet Learning)

Le projet répondant à cette question se nomme Learning.sln. Une fois lancé, le formulaire suivant s'affiche :

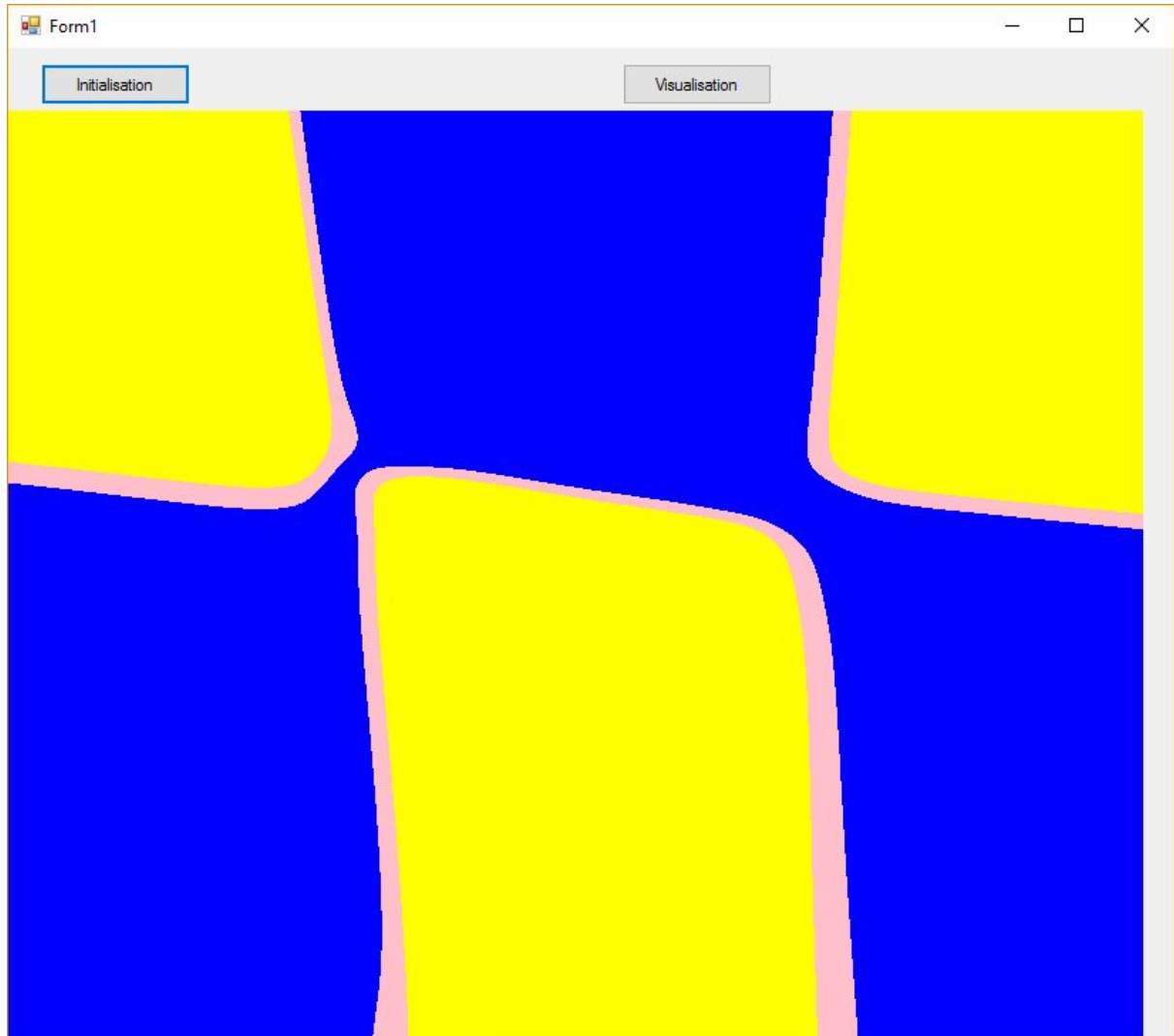


Apprentissage supervisé

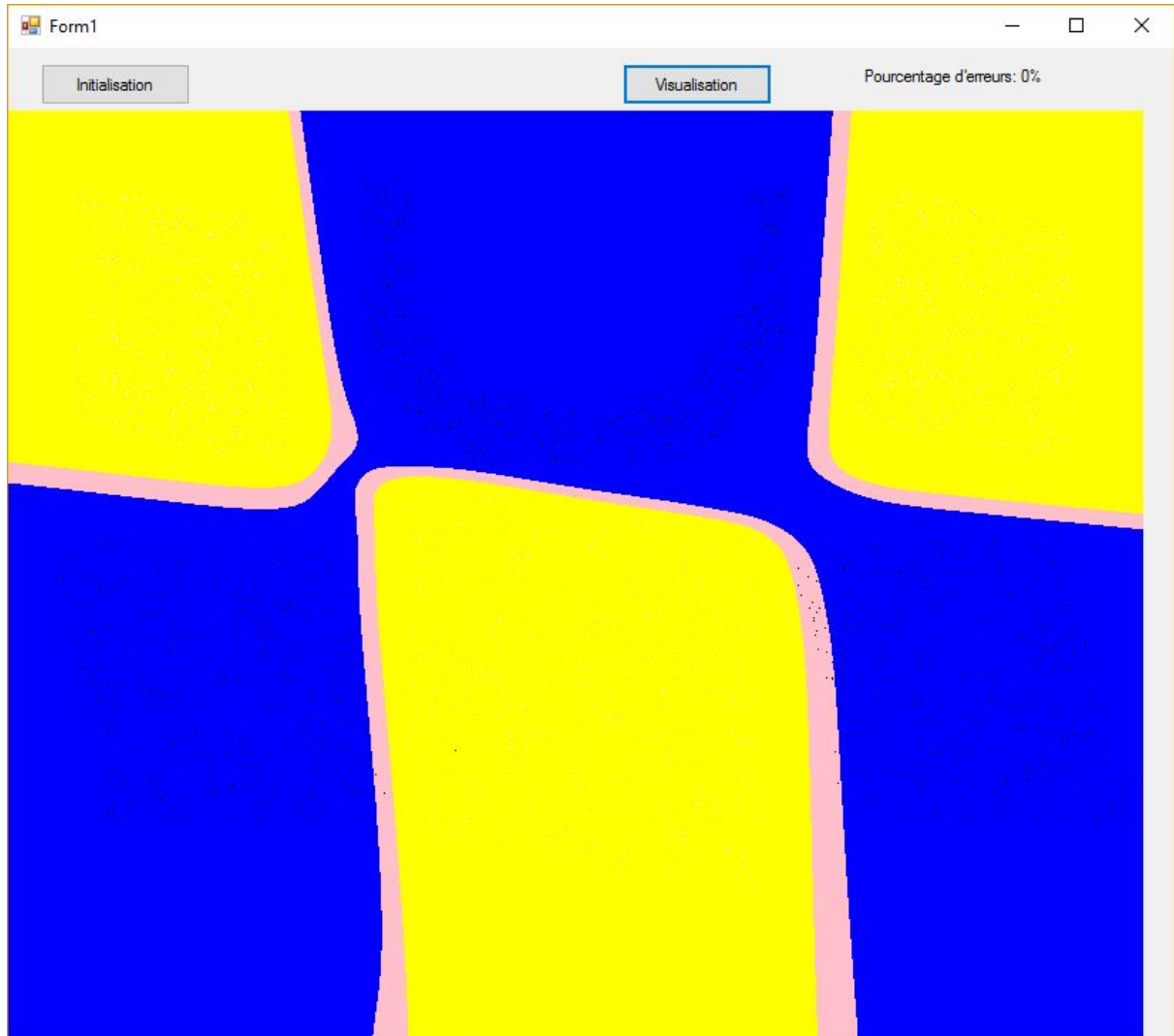
En cliquant sur Apprentissage supervisé, le formulaire suivant apparaît :



En cliquant sur le bouton initialisation, l'apprentissage démarre automatiquement. Les zones en jaunes ainsi que les zones bleues présent sur le formulaire symbolisent la classification opérée par notre perceptron: les pixels en jaune sont catégorisés dans la classe A, tandis que les pixels bleus sont catégorisés dans la classe B. Cette classification s'effectue selon un perceptron multi couche. Lorsque la sortie est inférieure à 0,4, le pixel est colorié en jaune. Si elle est supérieure à 0,6 le pixel est colorié en bleu. Dans le cas où la sortie est comprise dans l'intervalle $]0,4;0,6[$, le pixel sera colorié en rose.

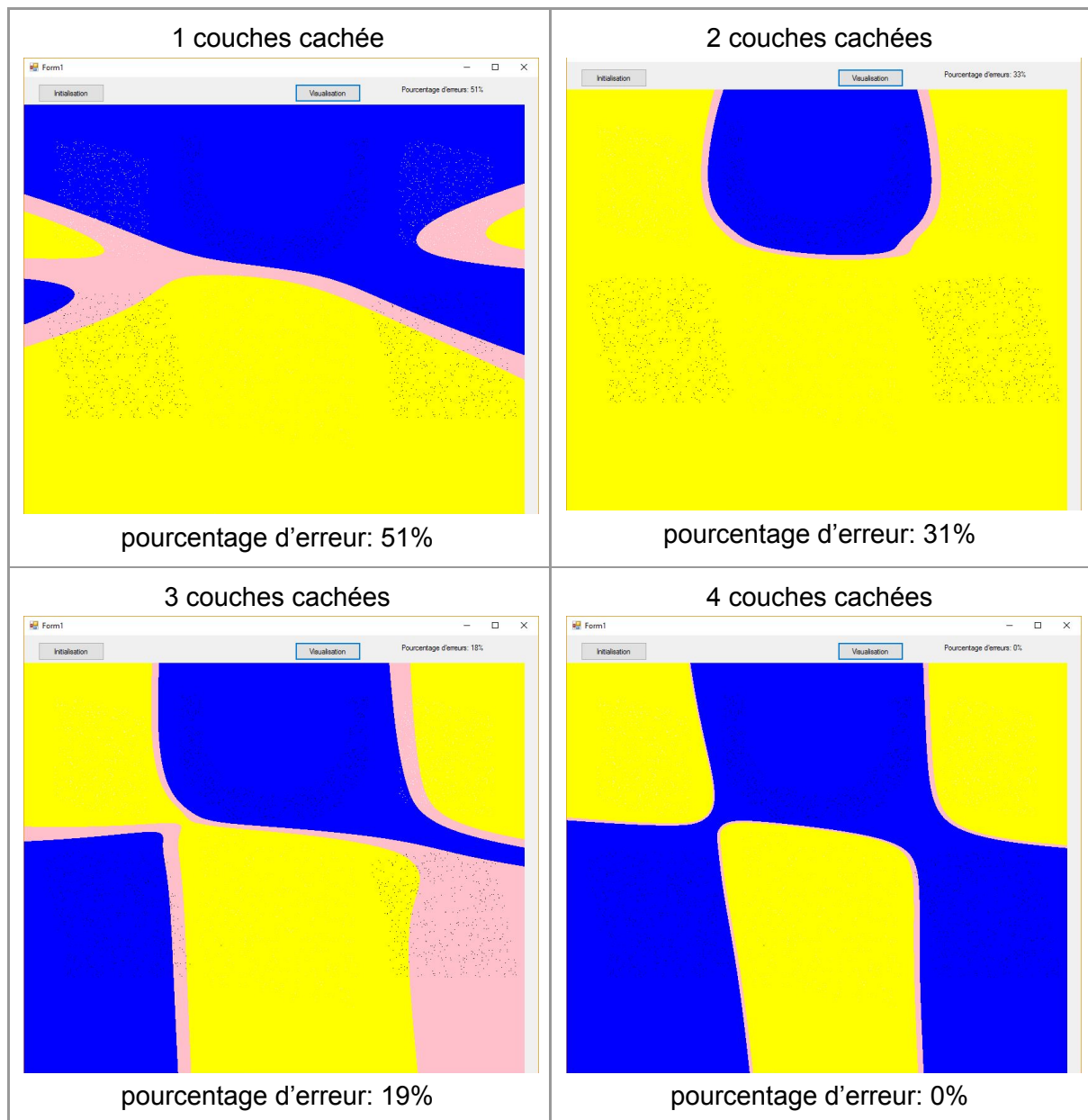


Le bouton *Visualisation* superpose des points représentant les données d'origine du fichier texte `datasetclassif.txt`. Les données d'origines appartenants à la classe A sont en blancs, tandis que les données d'origines appartenant à la classe B sont en noirs. Il permet également d'afficher le taux d'erreur en pourcentage de l'apprentissage du perceptron.



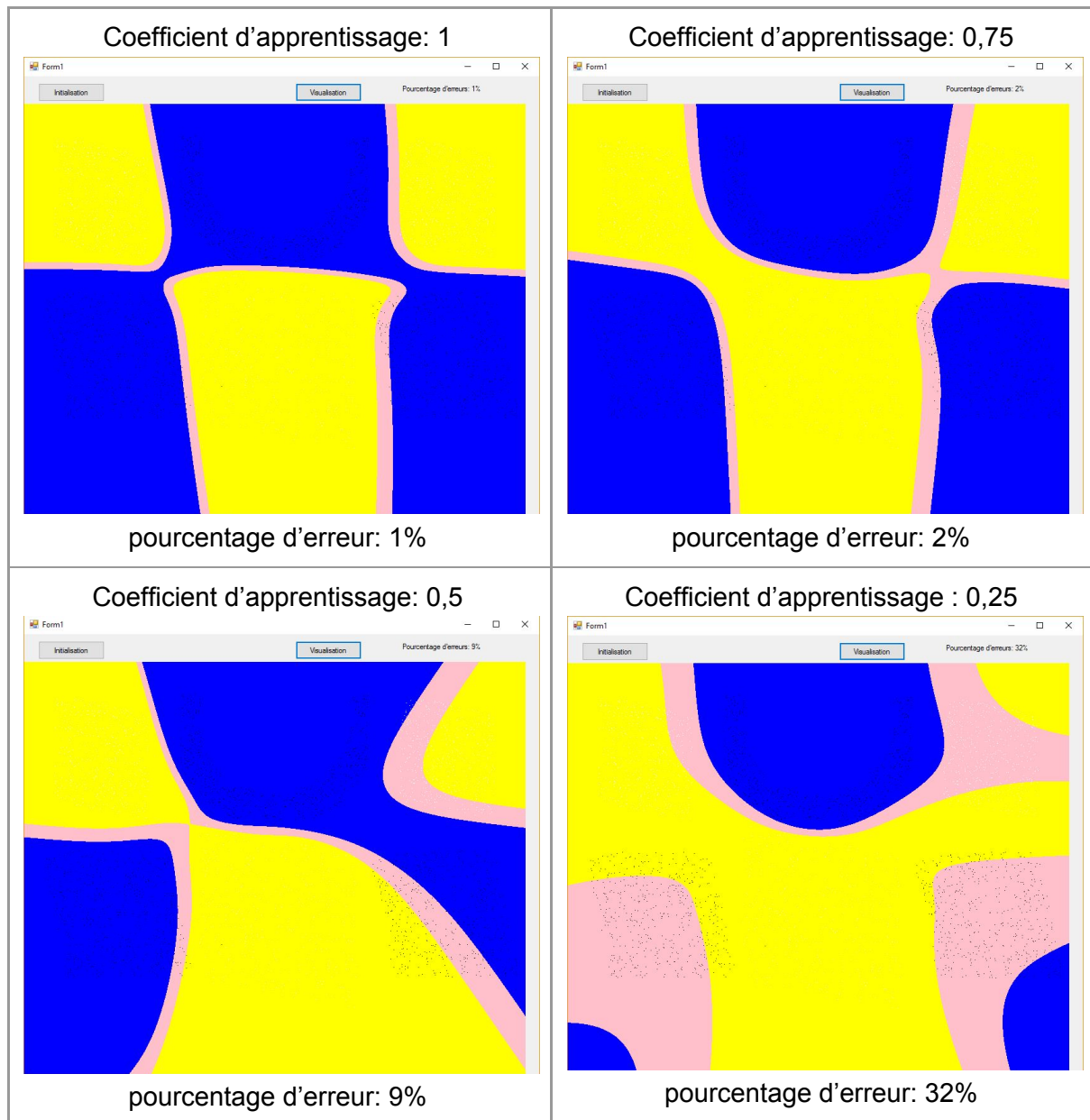
L'image ci-dessus illustre l'effet du bouton visualisation: superposition de points blancs et de points noirs sur l'image résultant de l'apprentissage du perceptron.

Influence du nombre de neurones sur la couche cachée

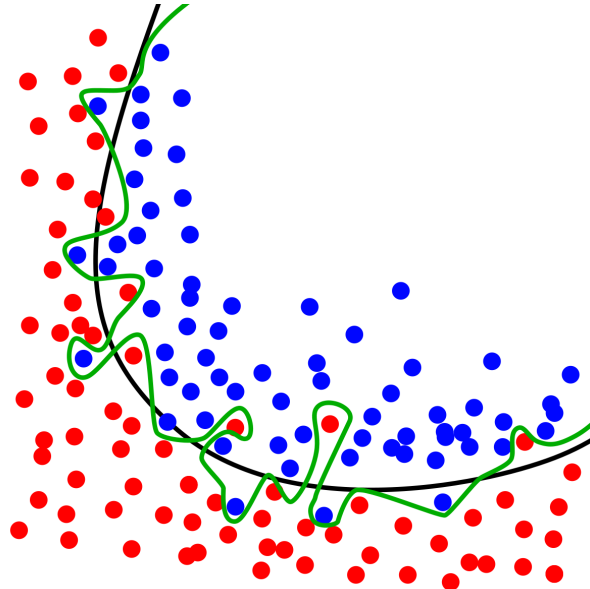


On remarque que la qualité et la vitesse de convergence augmentent avec le nombre de couches cachées. En effet, plus le nombre de couches cachées augmente, plus le nombre de traitements effectués augmente, ce qui permet d'améliorer le résultat. Cela ne s'applique cependant pas pour un nombre de couches cachées trop importants.

Influence du coefficient d'apprentissage (nombre de répétition 500)



On remarque que la qualité des résultats augmente avec le coefficient d'apprentissage. La vitesse de convergence ne varie pas selon le coefficient d'apprentissage. Il est à préciser que le nombre de répétition effectué ici est de 500, ce qui est un nombre pas suffisant pour compenser un faible coefficient d'apprentissage. Si le nombre de répétition est très important, on aurait assisté au phénomène inverse : un coefficient d'apprentissage faible aurait donné de meilleurs résultats, tandis qu'un coefficient d'apprentissage important aurait abouti à un phénomène d'overfitting: l'influence des erreurs de mesures sur l'apprentissage sera plus significative.



Influence du nombre de répétition

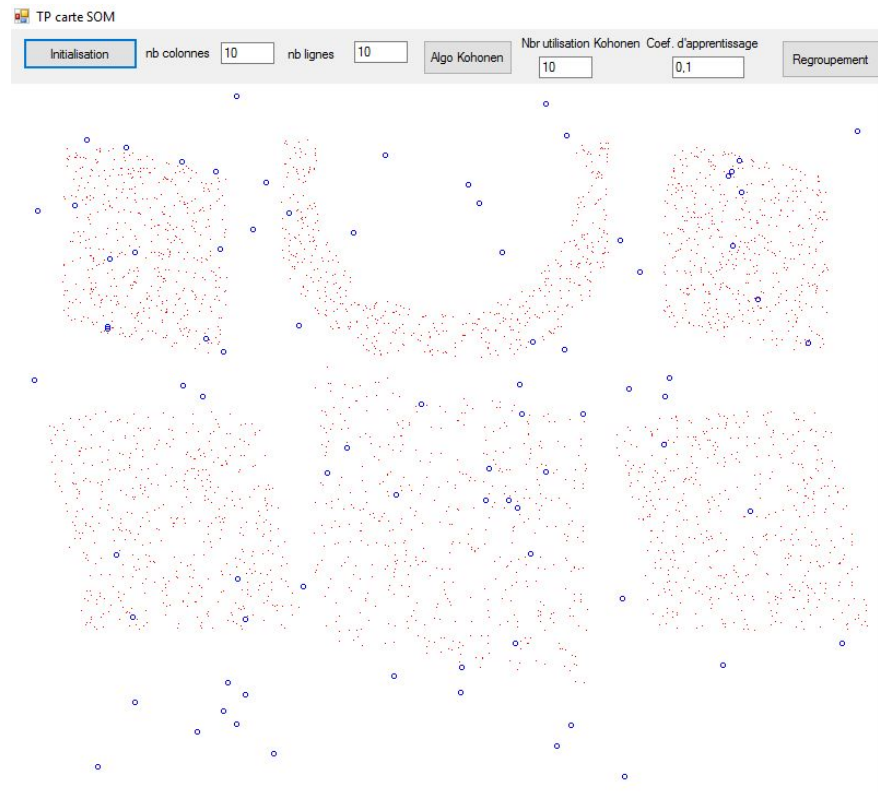
La qualité des résultats augmentent souvent avec le nombre de répétitions, au détriment de la vitesse de convergence. Cependant, nous avons parfois pu observer des cas d'overfitting lorsque celle-ci était trop importante.

Apprentissage non supervisé

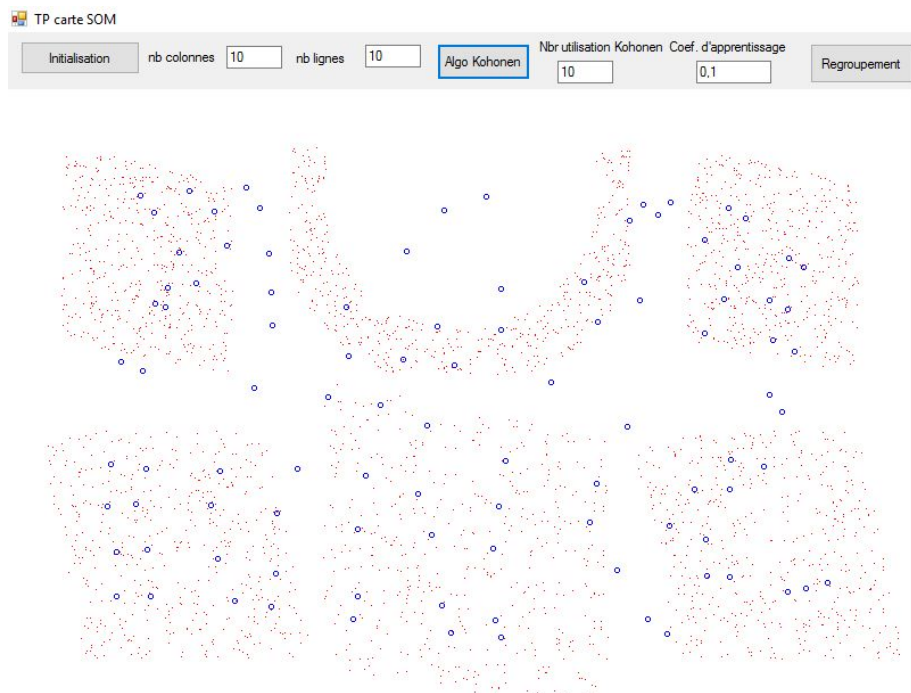
En cliquant sur Apprentissage supervisé, le formulaire suivant s'affiche:

A screenshot of a web form titled "TP carte SOM". The form has a light gray background and a white border. It contains several input fields and buttons. The first row has a blue "Initialisation" button, followed by "nb colonnes" (10), "nb lignes" (10), "Algo Kohonen", "Nbr utilisation Kohonen" (10), "Coef. d'apprentissage" (0,1), and a "Regroupement" button. The second row is empty. The form is titled "TP carte SOM" in the top left corner.

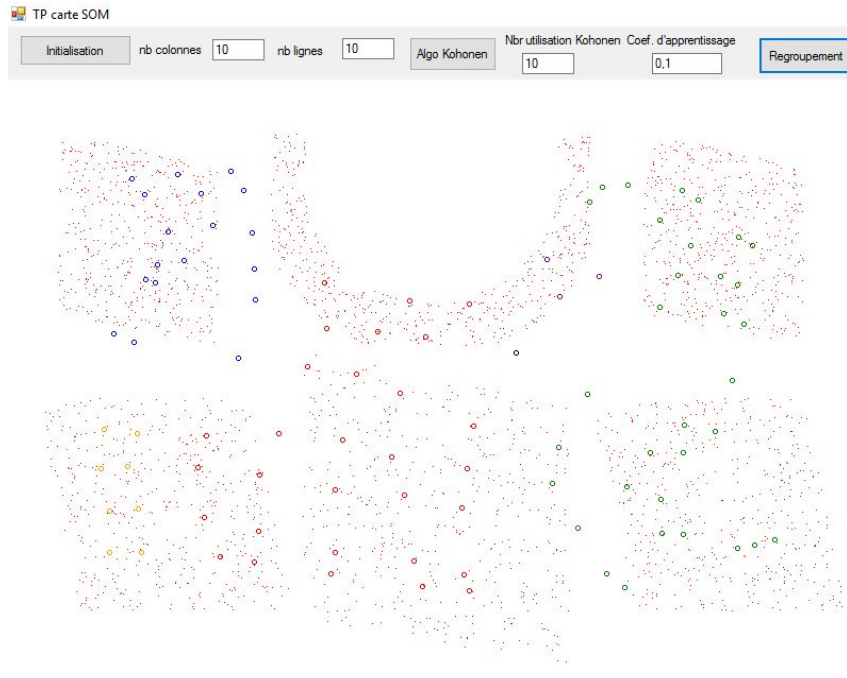
Ce formulaire permet de déterminer le nombre de ligne et de colonne de la carte qui va être affiché à l'écran, déterminant le nombre de neurone. Le paramètre Nbr utilisation Kohonen permet de déterminer le nombre de fois que l'algorithme de Kohonen sera utilisé lorsque l'on appuie sur le bouton Algo Kohoen. Une fois après avoir appuyé sur initialisation, la carte s'initialise:



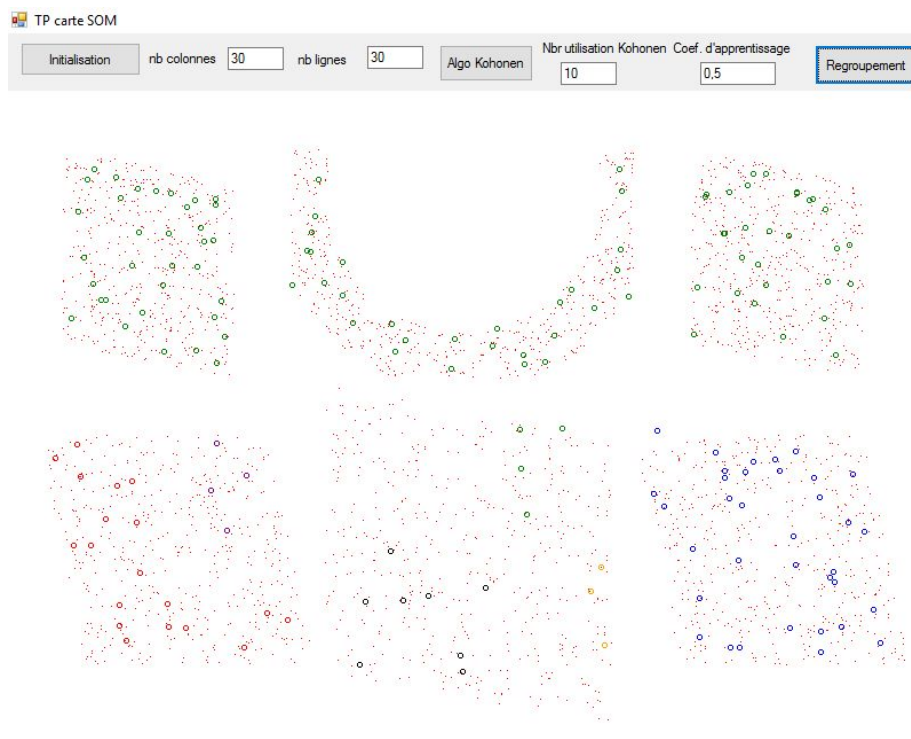
Il faut ensuite appuyer sur le bouton Algo Kohonen. Dans notre exemple, l'algorithme sera exécuté 10 fois.



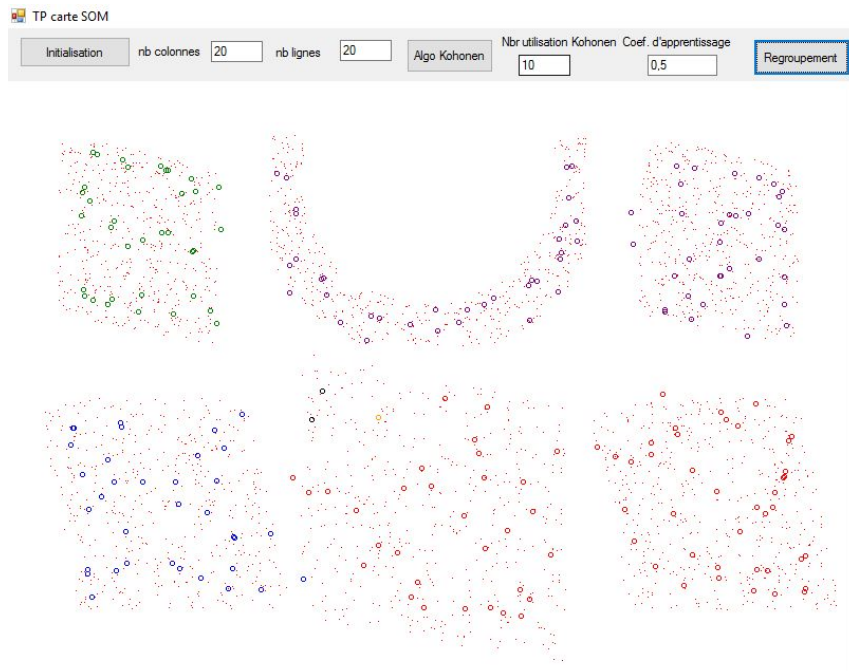
En appuyant sur le bouton Regroupement, les données seront triées par couleur selon la classe à laquelle elles appartiennent.



Influence du nombre de neurones de la carte:



Dimension 30*30



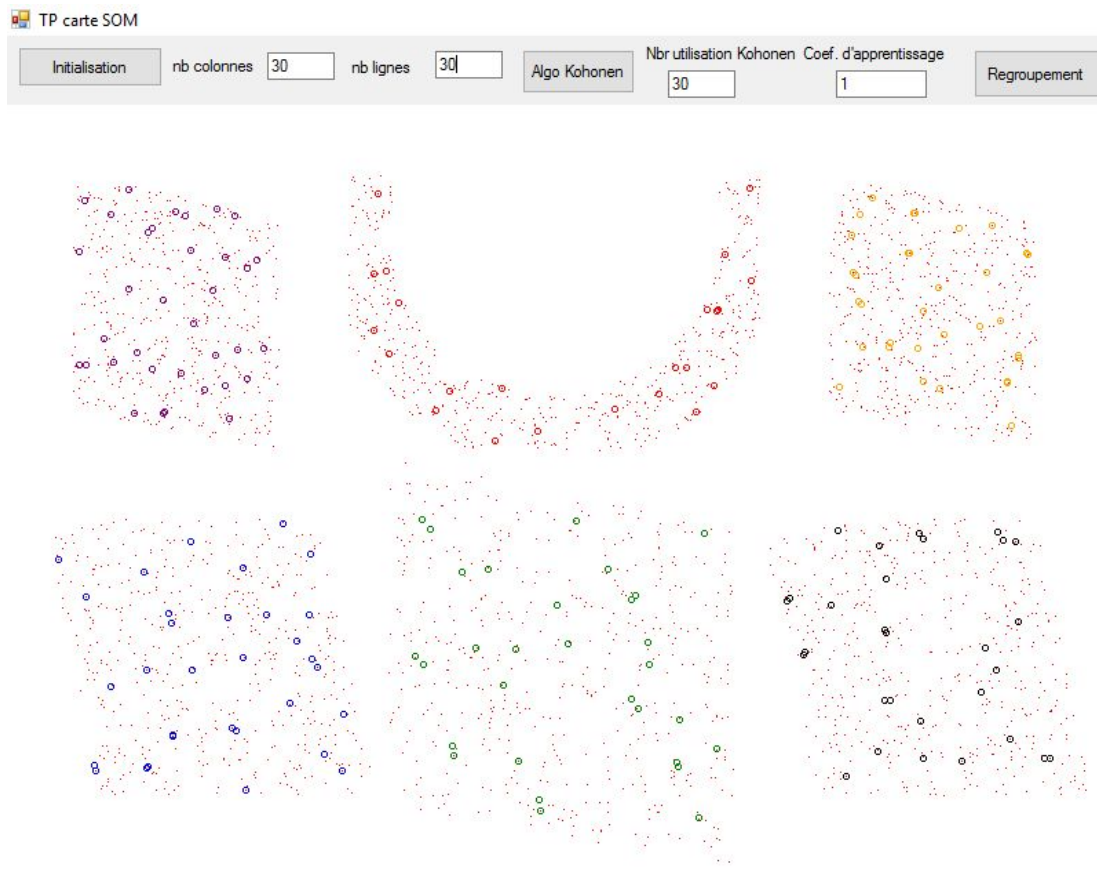
Dimension 20*20



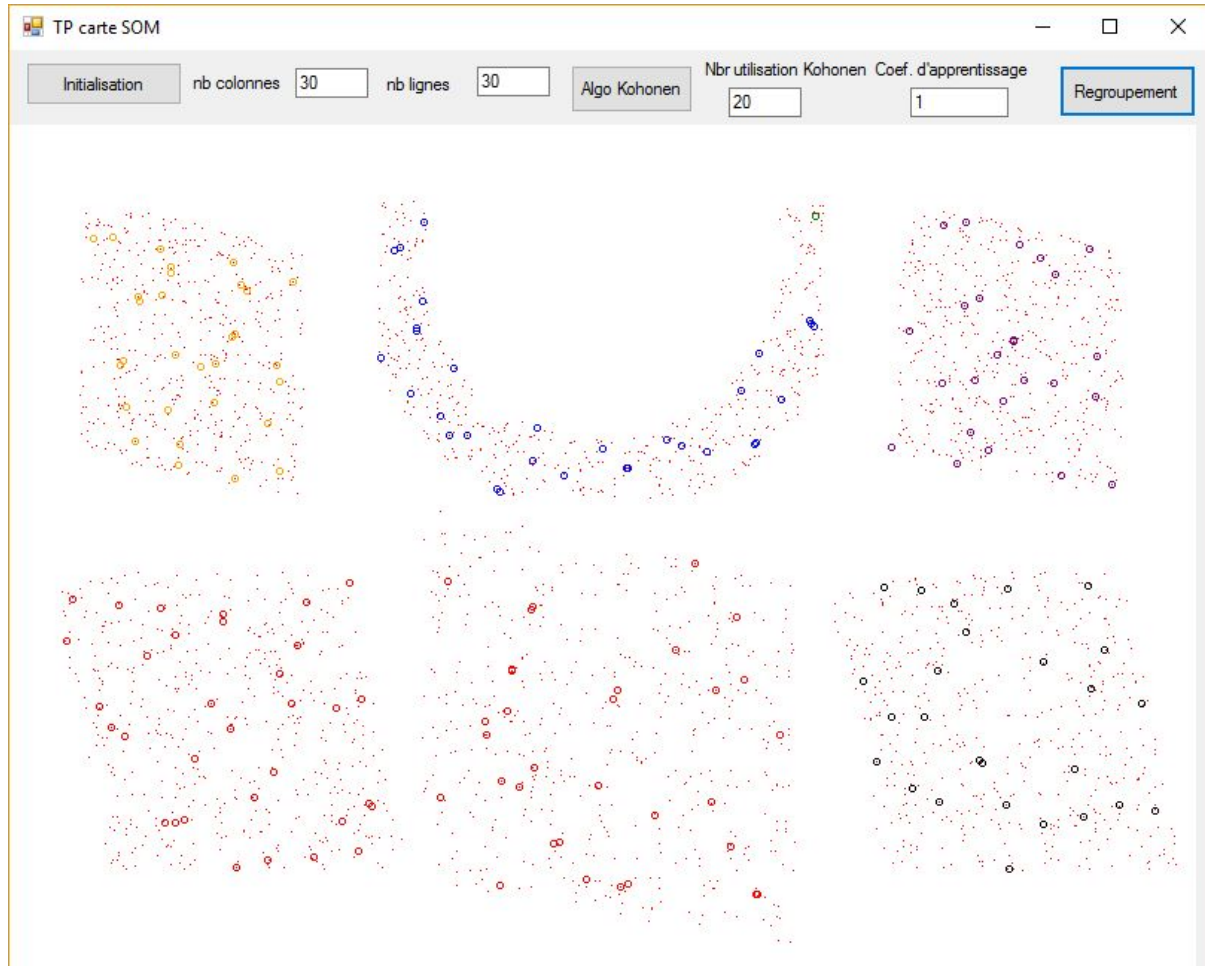
Dimension 10*10

On remarque que plus le nombre de neurones est important, plus les données sont réparties au sein des ensembles rouges, délimitant les différentes catégories des données du fichier texte. Cela s'explique par le fait que l'augmentation de neurones permet de générer plus de traitements visant à classifier les données, affinant ainsi les résultats obtenus.

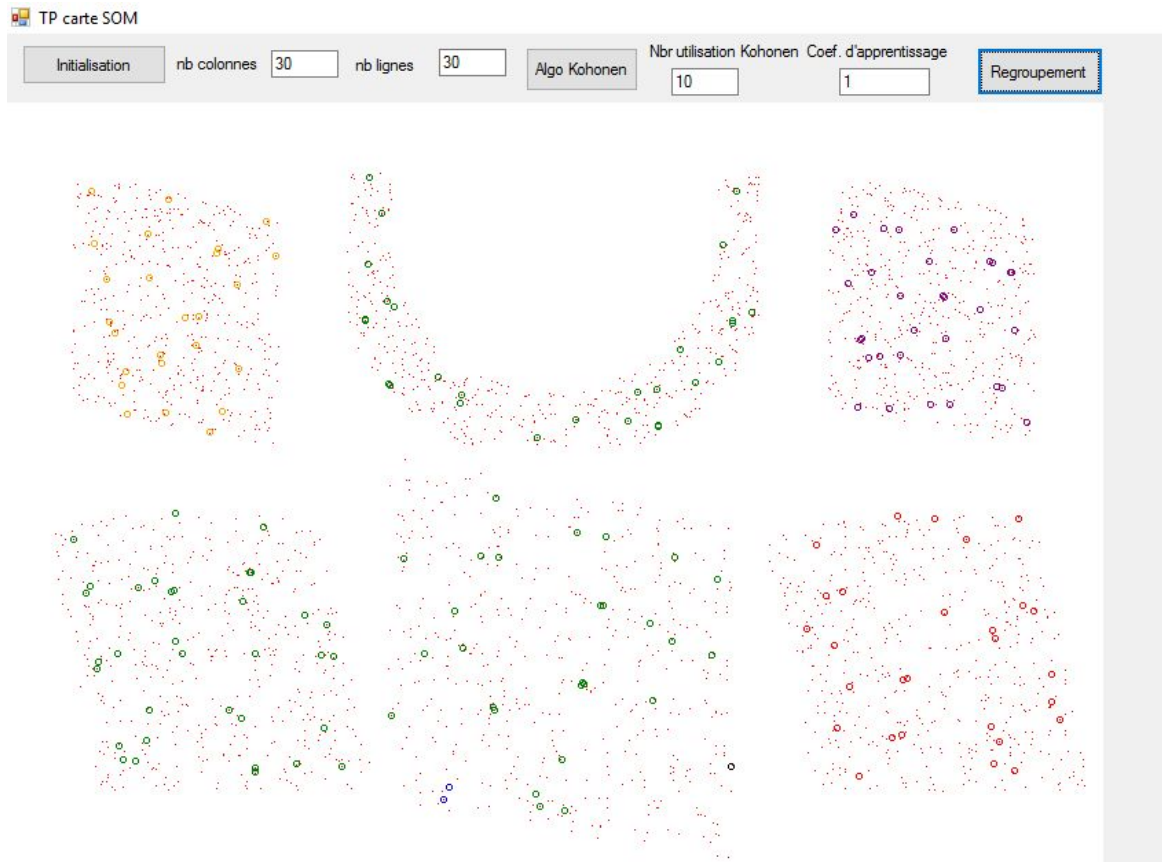
Influence de la vitesse de convergence:



Nombre d'utilisation de l'algorithme de kohohen : 30



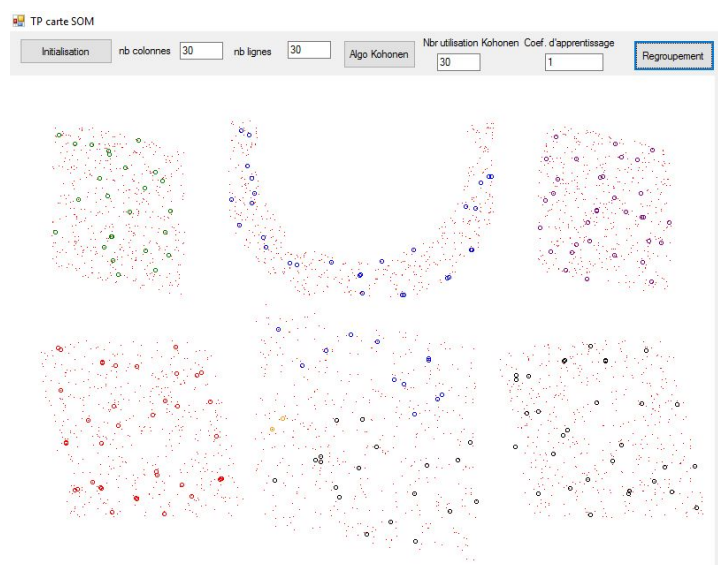
Nombre d'utilisation de l'algorithme de kohonen : 20



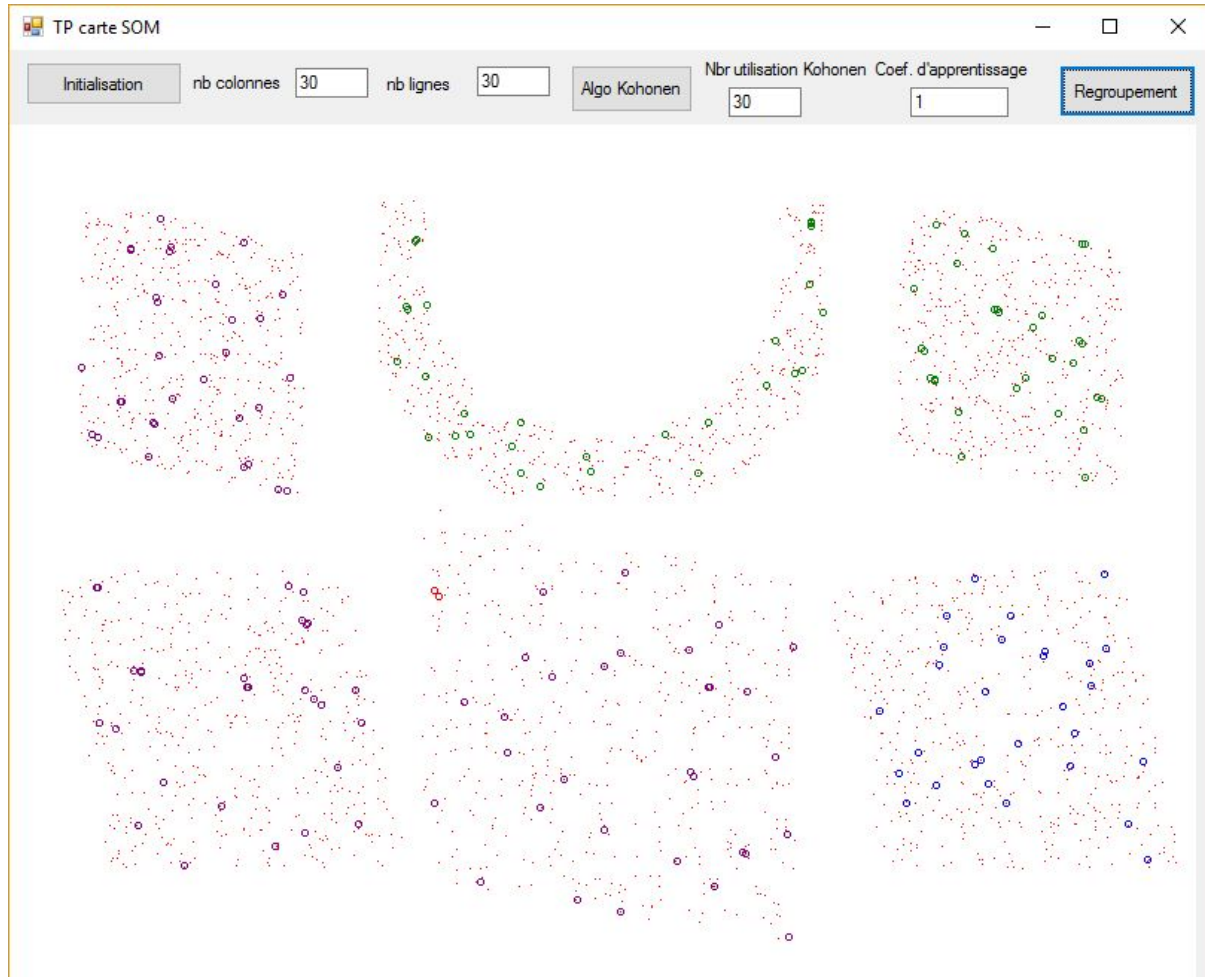
Nombre d'utilisation de l'algorithme de kohonen : 10

On remarque que plus l'algorithme est utilisé, mieux les données sont réparties dans les ensembles. Cela s'explique par le fait que plus l'algorithme est utilisé, mieux les poids de chaque neurone sont réparties, ce qui permet d'obtenir de meilleurs résultats.

Influence du nombre de classe demandé:



Nombre de classe: 6



Nombre de classe : 4



Nombre de classe : 2

Nous pouvons constater que plus le nombre de classe augmente, plus les données sont situés au coeur des ensemble et sont bien répartis au sein de leurs ensembles respectifs. Cela s'explique par le fait que l'augmentation du nombre de classe permet d'affiner le regroupement et le tri des données.

Conclusion

Ce projet fut pour nous l'occasion de mettre en pratique les enseignements que nous avons pu recevoir dans le module. Il nous a permis de nous familiariser davantage avec le langage C# en mettant en place des fonctions plus complexes que dans nos précédents projets, et d'avoir une expérience dans la mise en place et l'utilisation d'algorithmes utilisés afin de programmer une intelligence artificielle.