
Développement d'un simulateur de navigation pour un drone marin de surface avec un IA détecteur d'obstacles

Stage réalisé du 23 avril au 2 juin 2017, au sein du Laboratoire Informatique, Image et Interaction de l'Université de La Rochelle



Maxime GABORIT

Maitre de stage : Mr Michel MENARD

Enseignant réfèrent : Mr Renaud PETERI

REMERCIEMENTS

Dans un premier temps, je remercie l'équipe pédagogique de l'Université de La Rochelle qui, pendant ces trois années de Licence, m'aura permis d'acquérir les connaissances nécessaires pour réaliser ce stage.

Ensuite j'aimerais remercier le Laboratoire Informatique, Image et Interaction de m'avoir accueilli et permis de faire ce stage dans de bonnes conditions.

Enfin, je tiens à remercier mon maitre de stage, Mr Michel Menard, ainsi que Mr Bruno Lescalier de m'avoir guidé pendant toute la durée du stage. J'aimerais également les remercier pour les conseils, l'aide et toute la documentation qu'ils m'ont fournies pour me permettre de bien comprendre le sujet du stage.

TABLE DES MATIERES

Résumé du sujet.....	4
I- Le Laboratoire Informatique, Image et Interaction	5
1- Présentation du laboratoire	5
2- Historique	5
II- Cadre du stage.....	6
1- Structure	6
2- Espace de Travail	7
a- Matériel utilisé.....	7
b- Logiciels, environnement de travail, framework et technologies utilisées	7
III- Planning prévisionnel vs planning réel.....	9
IV- Travaux réalisés.....	10
1- Réalisation de l'interface graphique du simulateur	10
a- Description et fonctionnalités de l'interface.....	10
b- Le drone simulé et son sonar	12
c- Suivi du tracé par le drone.....	13
d- Simulation de bateaux.....	15
2- L'Intelligence artificielle.....	17
a- Qu'est-ce que le deep learning ?	17
b- Le réseau de neurones artificiels.....	18
c- Mon modèle de réseau de neurones	20
d- L'implémentation du réseau sur le drone	21
Conclusion	23
Annexes	24
Bibliographie / sitographie.....	25

Résumé du sujet

Durant ce stage, il m'était demandé de réaliser un simulateur pour un drone marin de surface. Le simulateur propose une vue du dessus du port des Minimes de La Rochelle sur laquelle il nous est possible de positionner des points de passage définissant la trajectoire à suivre du drone simulé. Le simulateur comprend aussi des bateaux qui suivent des trajets prédéfinis.

Le but étant que le drone suive le trajet défini tout en évitant les obstacles mobiles (bateaux) et immobiles (pontons, berges, ...). Cela se fait avec l'implémentation d'une Intelligence artificielle de type apprentissage par renforcement.

Le développement de ce simulateur s'est donc découpé en deux parties. La première étant la réalisation purement graphique du simulateur en langage python. La seconde était consacrée aux réseaux de neurones, je devais me familiariser avec cette technologie pour ensuite l'utiliser à l'intérieur de mon application.

Ce sujet de stage fait partie d'un projet qui consiste à améliorer une application iPad existante permettant de piloter le drone manuellement, afin d'y ajouter une fonctionnalité permettant de définir un trajet. A travers ce projet, le drone doit aussi, avec une caméra, détecter les obstacles et les éviter grâce à une intelligence artificielle.

Le but final de ce projet est donc de mettre en commun le travail de Guillaume Deau, qui développait l'application, celui de Mathieu Godignon qui s'occupait de la détection d'obstacles via la caméra et le mien, qui consistait au développement d'une intelligence artificielle, et d'implémenter ces travaux sur le drone Cyberjet.

I- Le Laboratoire Informatique, Image et Interaction

Ce stage s'est déroulé au sein du Laboratoire Informatique, Image et Interaction (L3i), voici une présentation du laboratoire ainsi qu'un historique de ses activités.

1- Présentation du laboratoire

Créé il y a 24 ans, le L3i est le laboratoire de recherche du domaine des sciences du numérique de l'Université de La Rochelle. Une centaine de personnes y travaille chaque jour, répartie entre l'IUT et le Pôle Sciences et Technologiques.

Parmi cette centaine de personnes, on peut retrouver 12 professeurs, 22 Maîtres de Conférences (dont 2 habilités à diriger des recherches), 8 Ingénieurs-secrétaires, 33 doctorants et 4 attachés temporaires d'Enseignement et de Recherche.

Le laboratoire se structure en 3 équipes, chacune se focalisant sur une thématique centrée sur la problématique de la gestion interactive et intelligente des contenus numériques. La première équipe travaille sur le thème « Modèles et connaissances », la deuxième se concentre sur la thématique des « Images et contenus » et enfin la dernière s'intéresse à la « Dynamique des systèmes et adaptativité ».

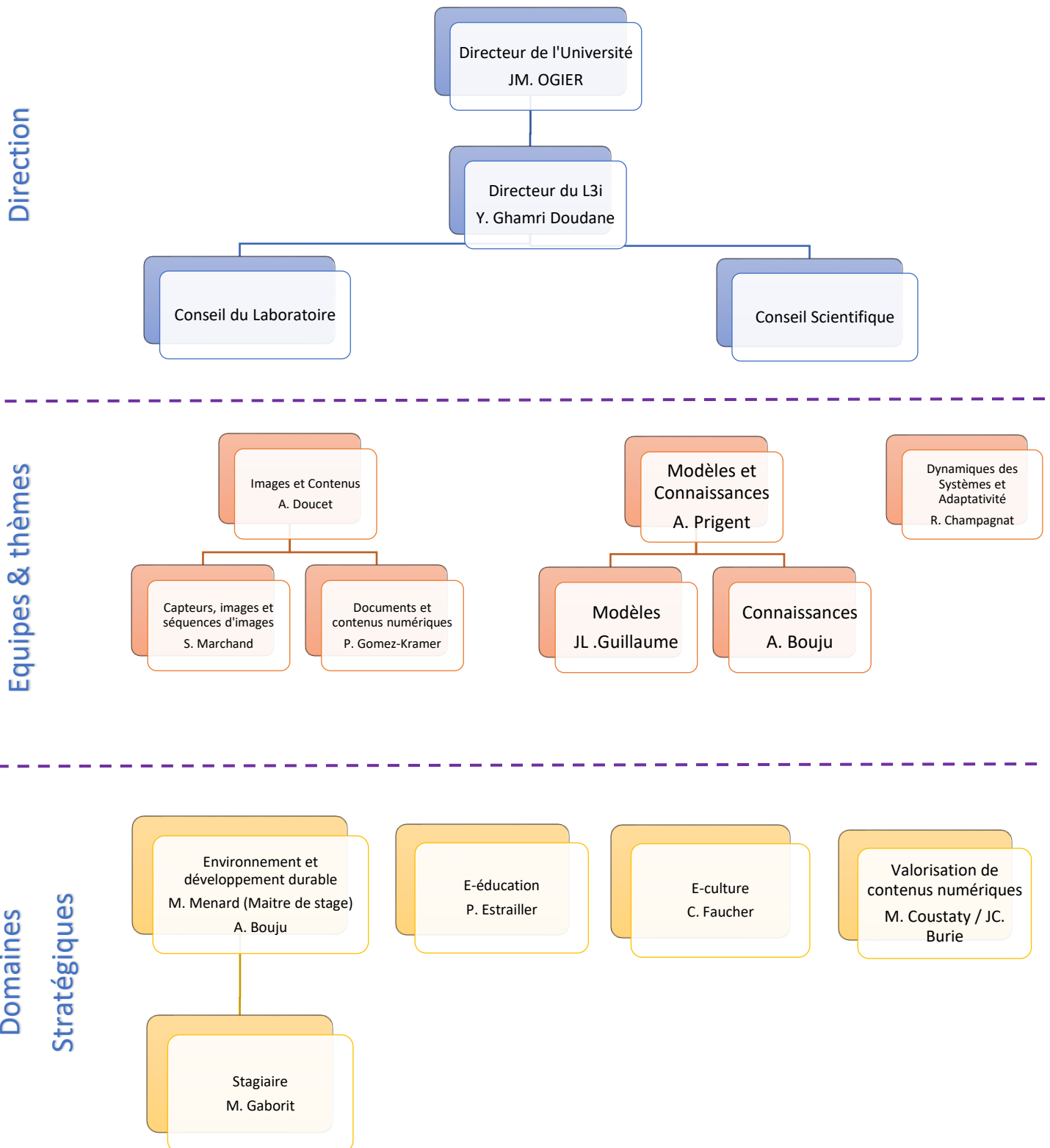
2- Historique

Quelques dates importantes :

- 1993 : Création du laboratoire Informatique et Imagerie Industrielle
- 1997 : Labellisation du laboratoire en Equipe d'Accueil du Ministère de la Recherche
- 2003 : Renommé en Laboratoire Informatique, Image et Interaction
- 2007 : Labellisation de l'équipe de recherche en Technologie « Interactivité Numérique »
- 2008 : Intégration dans le programme régional PRIDES
- 2011 : Obtention de la note A suite à l'évaluation de l'AERES

II- Cadre du stage

1- Structure



2- Espace de Travail

Mon stage s'est déroulé au sein du Laboratoire Informatique, Image et Interaction, sous la responsabilité de Mr Michel Ménard.

a- Matériel utilisé

Pour ce qui est du lieu du stage, je me suis installé en salle 133, au premier étage du bâtiment Pascal.

Pour réaliser ce simulateur, je n'avais besoin d'aucun matériel spécial, je l'ai donc réalisé sur mon mac book pro.

b- Logiciels, environnement de travail, framework et technologies utilisées



Afin de partager mon avancement avec Mr Ménard, j'ai créé un **git** où j'y déposais mon code le plus souvent possible. Ce git contient aussi une brève explication du projet, le diagramme de classe ainsi qu'un tutorial pour installer et tester le simulateur. Le code est disponible en cliquant [ici \[lien hypertext\]](#)



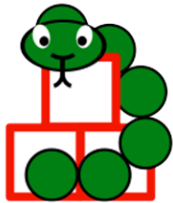
Suite au conseil de Mr Ménard, j'ai utilisé **Anaconda Navigator** pour créer un environnement de travail personnalisé. En effet, ce logiciel permet de créer des environnements sous différentes versions de python et y installer des packages et Framework facilement avec la commande « pip install ». Certaines librairies décrites ci-dessous n'étant pas disponibles ou peu stables sur la dernière version de python (3.6), j'ai donc évolué sur un environnement avec la version de python 2.7.



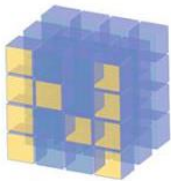
Les librairies et Framework étant installés sur l'environnement de travail anaconda, je n'ai donc pas utilisé d'IDE spécial pour python tels que *Pycharm* ou *Wing*, j'écrivais simplement mes fichiers python avec l'éditeur de texte **Sublime Text** et lançais le programme en ligne de commande.



Pour l’affichage graphique du simulateur j’ai utilisé **Pygame**, une bibliothèque libre construite sur la SDL (Simple DirectMedia Layer). Elle est à la base destinée à la création de jeux vidéo en python mais cette librairie se prêtait bien à la structure du simulateur.



Afin de ne pas m’occuper manuellement des déplacements, rotations et collisions des différents éléments présents à l’intérieur du simulateur (drone et bateaux simulés), j’ai donc installé le moteur physique 2D **Pymunk**, très simple à utiliser, multi-plateformes, et le plus important, compatible avec Pygame. L’autre avantage en plus de sa simplicité d’utilisation est que malgré son ancienneté (10 ans), cette bibliothèque est toujours mise à jour et très bien documentée.



Afin de réaliser l’Intelligence artificielle et son réseau de neurones, j’avais besoin d’une bibliothèque capable de créer des tableaux et effectuer des opérations sur ceux-là, des opérations un peu plus complexes que ce que peut offrir la librairie *math* incluse à l’installation de python. **Numpy** s’est avérée être la plus populaire pour ce type de manipulations et elle est surtout utilisée avec la bibliothèque que j’ai choisie pour réaliser le réseau de neurones.



La dernière librairie utilisée est **TensorFlow**, elle est libre et open source, développée par Google et permet de construire et entrainer un réseau de neurones.

III- Planning prévisionnel vs planning réel

Planning prévisionnel		semaine 1					semaine 2					semaine 3					semaine 4					semaine 5					semaine 6				
		du 24/04 au 28/04					du 01/05 au 05/05					du 08/05 au 12/05					du 15/05 au 19/05					du 22/05 au 26/05					du 29/05 au 02/06				
Tâches	Durée en jours	L	M	M	J	V	L	M	M	J	V	L	M	M	J	V	L	M	M	J	V	L	M	M	J	V	L	M	M	J	V
Pré requis																															
Prise en main du sujet	1																														
Recherche sur le Q-learning et neural network (test de code)	2																														
Choix et installation des libraires/framework	1																														
Apprentissage de python/ pygame /pymunk	2																														
L'IHM du simulateur																															
Première fenêtre Pygame (image de fond, boucle d'événements)	1																														
Ajout de markers et du tracé	1																														
Implémentation du drone qui suit le trajet	2																														
Implémentation du sonar	2																														
Ajout de bateaux simulés	2																														
Intelligence artificiel																															
Prise en main de la librairie Tensorflow	3																														
Création du neural network	2																														
Implémentation du réseau de neurone sur le drone simulé	4																														
Phase de test	1																														

Figure 1 - Planning prévisionnel

Planning réel		semaine 1 du 24/04 au 28/04					semaine 2 du 01/05 au 05/05					semaine 3 du 08/05 au 12/05					semaine 4 du 15/05 au 19/05					semaine 5 du 22/05 au 26/05					semaine 6 du 29/05 au 02/06				
Tâches	Durée en jours	L	M	M	J	V	L	M	M	J	V	L	M	M	J	V	L	M	M	J	V	L	M	M	J	V	L	M	M	J	V
Pré requis																															
Prise en main du sujet	1																														
Recherche sur le Q-learning et neural network (test de code)	2																														
Choix et installation des librairies/framework	1																														
Apprentissage de python/ pygame /pymunk	2																														
L'IHM du simulateur																															
Première fenêtre Pygame (image de fond, boucle d'événements)	1																														
Ajout de markers et du tracé	1																														
Implémentation du body pymunk qui suit le trajet	2																														
Implémentation du sonar (translation et rotation avec le drone)	2																														
Amélioration des déplacement du drone (rotation et vitesse)	2																														
Ajout de bateaux simulés	2																														
Intelligence artificiel																															
Prise en main de la librairie Tensorflow	3																														
Création du neural network	2																														
Implémentation du réseau de neurone sur le drone simulé	4																														
Phase de test	1																														

Figure 2 - Planning réel

La prise en main du sujet a été plus courte que prévu, j'ai pu commencer à me plonger dans la création de l'interface dès la première semaine et ainsi la finir à la fin de la première semaine. Cela m'a permis de consacrer 3 semaines entières sur l'intelligence artificielle, même si au final, cela aura été un peu court.

IV- Travaux réalisés

Durant ces six semaines, mon travail sur le simulateur a été basiquement divisé en deux. La première partie consistait à la fabrication de l'interface graphique et la seconde à la création de l'intelligence artificielle.

1- Réalisation de l'interface graphique du simulateur

L'intégralité de cette interface a été écrite en python avec une utilisation des librairies *Pygame*, gérant la partie graphique et *Pymunk*, s'occupant de la partie physique du simulateur.

a- Description et fonctionnalités de l'interface

L'interface du simulateur possède deux boutons en bas à droite de la fenêtre et l'image de fond représente une vue du dessus du port des Minimes de La Rochelle. C'est une interface très simpliste mais après réflexion, concevoir une IHM ergonomique et visuellement élégante n'était pas le but principal de mon stage, j'ai donc jugé qu'il ne fallait pas que j'y passe trop de temps.

La bibliothèque *Pygame* ne contient malheureusement pas d'objet « Button » ou « menu » et la combiner avec une autre librairie spécialisée dans les interfaces comme *Tkinter* s'avère assez laborieux. Les boutons sont donc de simples rectangles, on détecte le clic grâce à une fonction de collision aux coordonnées de la souris et du rectangle. Il y a bouton « start/stop » qui permet de lancer et d'arrêter la simulation et un bouton « clear » qui supprime tous les points de passage.

Au lancement du simulateur, un marker rouge est placé sur la carte, ce sera notre point de départ. On peut ensuite, avec un clic gauche sur la carte, positionner un marker bleu qui est un point de passage pour le drone, une droite est tracée au fur et à mesure qu'on ajoute des markers. On peut également, avec un clic droit sur un marker, le supprimer. Le trajet s'actualise à chaque suppression d'un point de passage.

Pygame utilise une boucle d'évènements pour gérer les clics souris ou les entrées clavier, afin d'ajouter ou supprimer un élément de la fenêtre il faut donc entièrement la redessiner. J'ai donc défini 3 méthodes « redraw » :

- `redraw_empty()` : redessine le fond et les deux boutons
- `redraw()` : `redraw_empty()` + les markers et les lignes entre ceux-ci
- `redraw_simulation()` : `redraw_empty()` + les bateaux simulés et le drone

Ci-dessous, les captures d'écrans des résultats obtenus pour cette première partie.

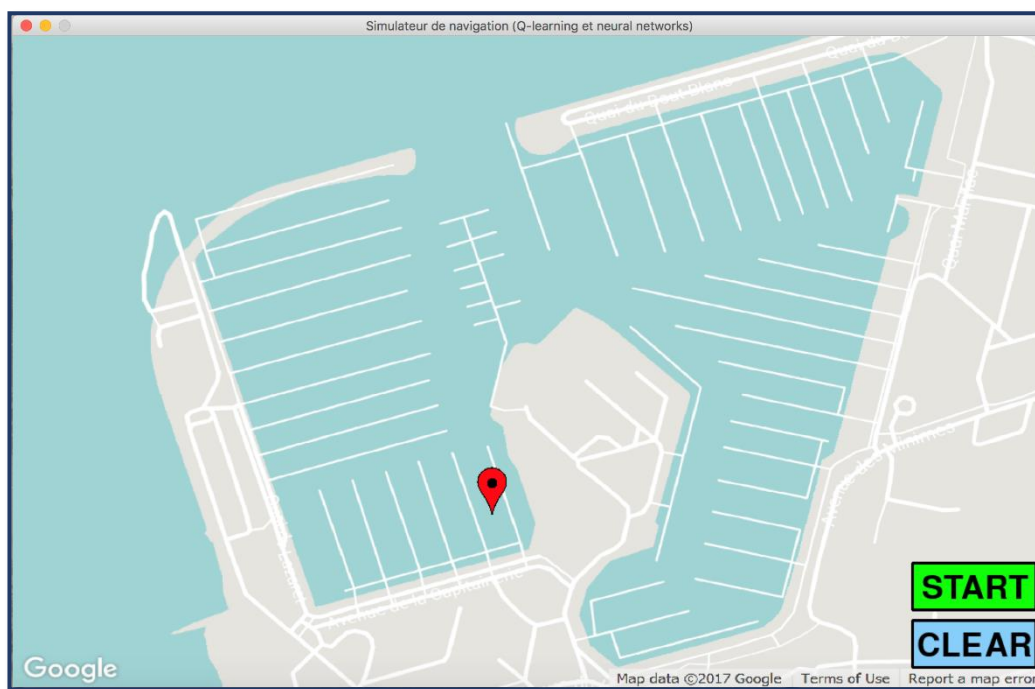


Figure 3 -Capture d'écran de la fenêtre pygame de départ

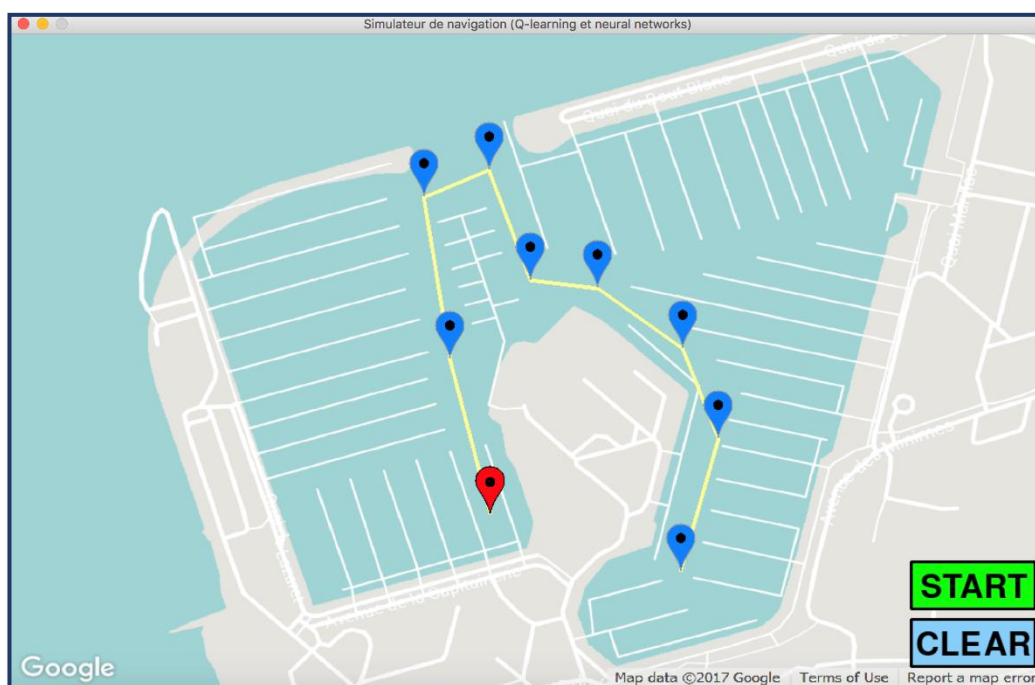


Figure 4 – Capture d'écran de la fenêtre avec des markers et le trajet

b- Le drone simulé et son sonar

L'implémentation du drone simulé dans la fenêtre *Pygame* s'est faite grâce à la bibliothèque *Pymunk*. Dans un premier temps j'ai dû créer un « space » qui est un espace, en plus de la fenêtre d'affichage, dans lequel on peut ajouter nos objets physiques. L'avantage de *Pymunk* est qu'il s'occupe des calculs mais aussi de l'affichage graphique. Maintenant nous avons donc deux couches graphiques, une avec le fond, les boutons, le trajet et la seconde avec les objets physiques.

Le drone en lui-même est un « rigid body » auquel on peut donner une forme, ici j'ai choisi le cercle, une taille, une position dans l'espace, une masse, une vitesse et beaucoup d'autres paramètres.

Il fallut maintenant détecter les obstacles se situant devant le drone, pour cela j'ai suivi l'exemple donné par Mr Ménard (lien en annexe) où la détection se fait grâce à un sonar composé de 3 bras, chacun discrétisé en 10 points de collision.

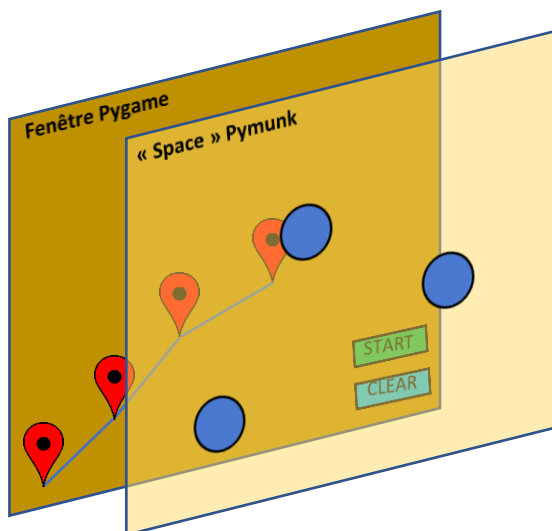


Figure 6 - Superposition de la fenêtre Pygame et de l'espace Pymunk

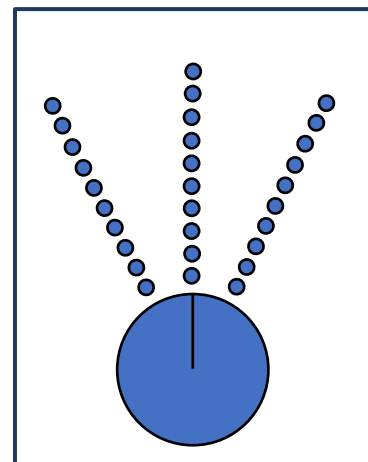


Figure 5 - Le drone et son sonar

Les points de collision du sonar ne sont pas des « body » *Pymunk*, il s'agit simplement de cercles que l'on dessine dans la fenêtre. Pour détecter une collision, le plus simple et le plus efficace revenait à comparer la couleur du pixel aux coordonnées des points du sonar à la couleur de l'eau. Si le pixel n'est pas bleu-turquoise, alors le sonar est entré en collision avec un bateau ou un ponton. Si collision il y a, on n'affiche pas les points situés après.

c- Suivi du tracé par le drone

Désormais, au clic sur le bouton start, le trajet et les markers s'enlèvent mais le drone et son sonar s'affichent aux coordonnées du point de départ. Il faut maintenant que celui-ci suive le trajet. Dans un premier temps, on va orienter le « rigid body » dans la direction du premier point de passage. Pour cela on calcule l'angle entre la position du drone et le prochain « waypoint » à l'aide la formule suivante :

$$\text{Angle} = \arctan2(y_{\text{waypoint}} - y_{\text{drone}}, x_{\text{waypoint}} - x_{\text{drone}})$$

L'objectif maintenant est de faire avancer le drone selon cet angle, cela est très facile avec la bibliothèque *Pymunk*. Chaque « rigid body » possède une vélocité qui résulte de la multiplication entre une vitesse et une direction, la direction étant calculée à partir de l'angle. On actualise donc la vitesse du « body » avec un entier supérieur à 0, ainsi le drone sera en mouvement jusqu'à ce qu'on remette la vélocité avec une vitesse de 0.

A chaque itération, gérée par une horloge de la librairie *Pygame*, il faut vérifier que le drone est arrivé au point passage, tant qu'il n'a pas atteint l'objectif, on garde le même angle. Quand les coordonnées du drone et celles du point de passage concordent, on passe au « waypoint » suivant. Cela veut dire qu'un nouveau calcul d'angle et une actualisation de la vélocité sont nécessaires. Dans le cas où il n'existe pas de prochain point de passage, cela signifie qu'on est arrivé à destination, la vitesse est alors mise à 0, la simulation est arrêtée et les « waypoint » réaffichés.

Cette méthode est efficace mais la direction du drone change brusquement et la vitesse n'est pas ajustée en fonction de l'angle. Etant en train de créer un simulateur, les déplacements du drone devaient donc se faire dans un esprit de simulation. Pour cela j'ai commencé par changer l'angle progressivement, celui-ci étant en radian, je l'augmente ou le diminue de 0.1 en fonction de la direction (bâbord ou tribord). Tant que le nouvel angle calculé n'est pas atteint on continue à changer progressivement la direction du drone.

La deuxième tâche à effectuer était d'ajuster la vitesse en fonction de l'angle du virage, par exemple si le virage est en épingle, on décélère fortement. On déclenche cet ajustement de vitesse quand on approche à 50 pixels du point de passage. Ci-dessous, un tableau de l'accélération ou la décélération en fonction de la situation du drone :

Situation du drone	Accélération (+) ou décélération (-)
Distance avant la fin entre 20 et 100 pixels	Vitesse - 10
Distance jusqu'au waypoint < 50 & angle < 45°	Vitesse - 0
Distance jusqu'au waypoint < 50 & angle < 90°	Vitesse - 5
Distance jusqu'au waypoint < 50 & angle < 135°	Vitesse - 10
Distance jusqu'au waypoint < 50 & angle > 135°	Vitesse - 15
Distance du waypoint précédent > 10	Vitesse + 20

En résumé, à chaque itération on appelle la fonction « `adjust_speed` » qui analyse la position du drone et ajuste sa vitesse. Si on approche de la destination, on décélère de -10, si le drone est à 50 pixels du point de passage, l'angle du virage déterminera la valeur de décélération (entre 0 et -15). Pour finir, si le virage a été effectué et que le drone se situe à 20 pixels du « waypoint » précédent, le drone va accélérer de + 20 unités par itération jusqu'à atteindre sa vitesse maximale qui est de 300.

Toutes ces méthodes de création (body *Pymunk* et sonar), de gestion des mouvements et de la vitesse se trouvent dans la classe « `boat` ». Cette classe sera réutilisée par la suite pour ajouter des obstacles mobiles au simulateur.

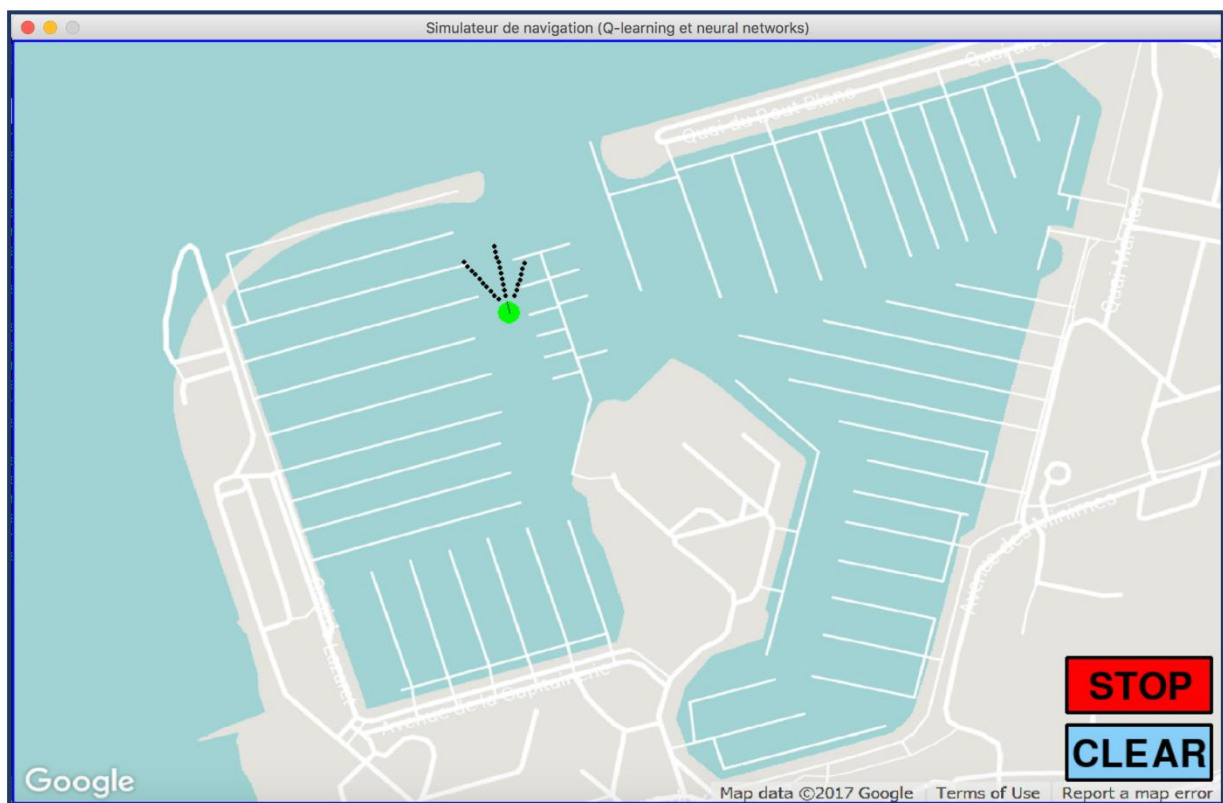


Figure 7 – Capture d'écran avec la simulation du drone et son sonar

d- Simulation de bateaux

La dernière partie du simulateur a été l'ajout d'obstacles mobiles, ce sont comme les drones des « rigid body » *Pymunk* de forme circulaire.

Le fonctionnement de ces bateaux simulés est le suivant : 12 trajets sont définis au départ, au lancement de la simulation 5 sont choisis aléatoirement, dès qu'un trajet se finit, un nouveau est tiré aléatoirement parmi les trajets restants. Il est possible d'augmenter ou diminuer le nombre de trajets ainsi que le nombre de bateaux affichés sur le simulateur.

La mise en place de ces bateaux simulés se découpe en trois classes :

- « Init_trajet_boat » :

Cette classe possède 12 méthodes nommées de `init_trajet_1` à `init_trajet_12`, chacune renvoyant une liste de waypoint. Il est tout à fait possible d'ajouter des trajets.

- « Trajet_boat » :

Cette deuxième classe s'occupe des mouvements du bateau simulé. Elle possède une liste de « waypoint » correspondant à un trajet de la classe « Init_trajet_boat » et un objet « boat » afin d'utiliser les méthodes de mouvements de cette classe. Chaque objet « boat » est défini avec un diamètre aléatoire entre 5 et 10 ainsi qu'une vitesse aléatoire entre 100 et 200.

- « Simulation_bateau » :

Cette dernière méthode est en charge du choix des bateaux simulés à afficher, elle possède deux listes : « list_trajet » et « trajet_en_cours ». Elle possède aussi les fonctions « start », « stop » et « run ». La fonction « start » est appelée au clic sur le bouton « start » et transfère aléatoirement 5 trajets de la liste « list_trajet » vers la liste « trajet_en_cours » et d'ajouter les « rigid body » dans l'espace *Pymunk*. La méthode « stop » fait le transfert, inverse et retire les bateaux de l'espace. Quant à la méthode run, elle s'occupe de déplacer les objets « boat » des 5 « trajet_boat » et dès qu'un trajet est complété, elle en sélectionne un nouveau.

Au lancement du simulateur je m'étais rendu compte que le sonar ne détectait pas les bateaux simulés. En effet, comme expliqué précédemment, la fenêtre *Pygame* et l'espace *Pymunk* sont totalement indépendants. La détection de la couleur du pixel ne fonctionne donc pas sur cet espace. J'ai dû ajouter une fonction « *draw_circle* » qui dessine un cercle, de la librairie *Pygame*, dans la fenêtre avec la même position et le même diamètre que le « *body* » *Pymunk*. Ce cercle est en dessous du « *rigid body* », il n'est donc pas visible mais permet au sonar de le détecter.

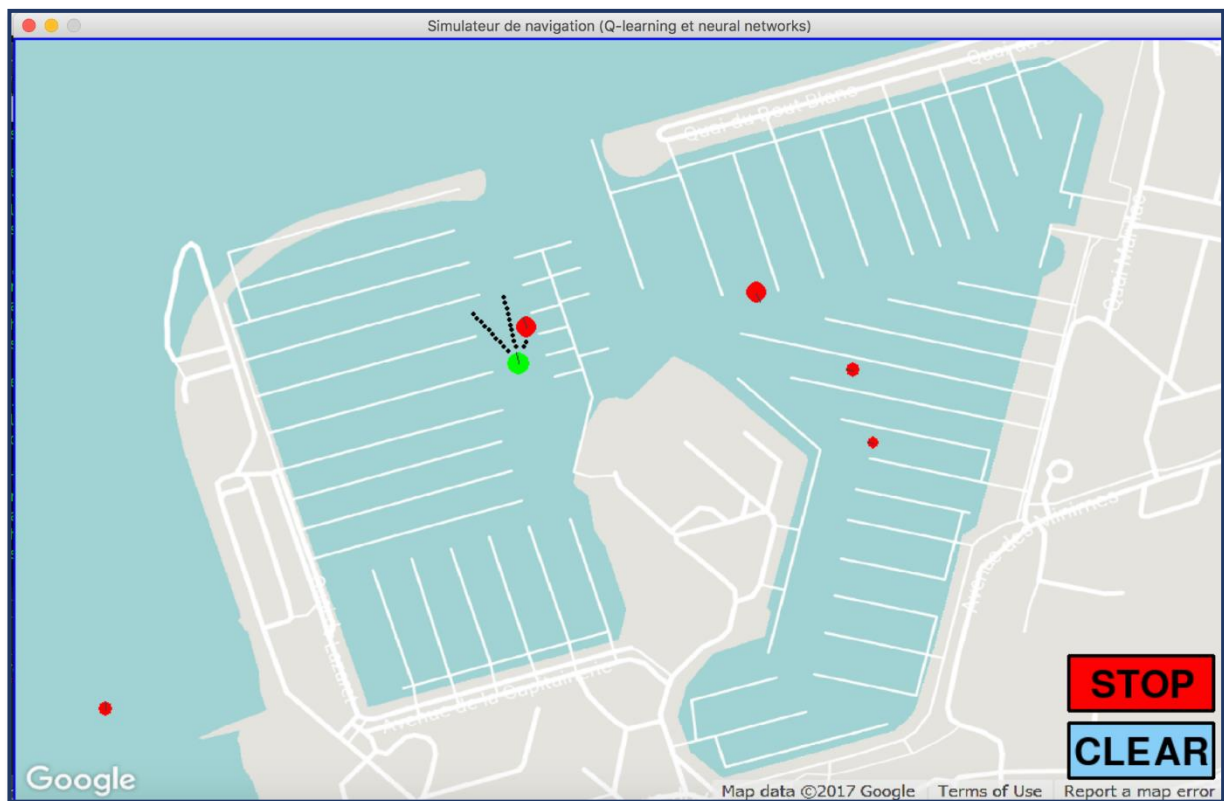


Figure 8 – Capture d'écran avec la simulation des bateaux

2- L'Intelligence artificielle

Le simulateur contient maintenant des bateaux simulés, un drone et un sonar qui détecte les obstacles fixes et mobiles. L'objectif est désormais d'éviter ces obstacles en continuant à suivre le chemin tracé. On va donc munir le drone d'une intelligence artificielle qui se va se baser sur le deep learning.

a- Qu'est-ce que le deep learning ?

Yann LeCun, chercheur en intelligence artificielle et considéré comme l'un des inventeurs du deep learning dit : « *il n'y a pas d'intelligence sans apprentissage, même les animaux ne possédant qu'une centaine de neurones apprennent.* ». Le machine learning recherche donc à entraîner un modèle à partir de données existantes pour qu'ensuite, ce même modèle puisse donner des prédictions, des tendances ou comportements futurs avec en entrée, de nouvelles données.

Ces modèles d'intelligence artificielle sont notamment utilisés sur le réseau social *Facebook*. A partir de plusieurs paramètres comme les informations personnelles, les commentaires et les likes de l'utilisateur, le machine learning peut prédire les pages et post que l'utilisateur peut potentiellement aimer et ainsi les proposer en page d'accueil. Plus l'utilisateur est actif sur le réseau social, plus l'algorithme aura de données pour s'entraîner et ainsi les prédictions seront plus précises et ciblées.

Le deep learning c'est une manière particulière de faire du machine learning. C'est une discipline qui a pris de l'ampleur au début des années 2010 lors d'une compétition de reconnaissance d'image où tous les meilleurs algorithmes du monde s'affrontent. Alors que la plupart des participants utilise des algorithmes conventionnels, c'est un modèle basé sur le deep learning qui s'impose. L'année suivante tout le monde a proposé un algorithme de deep learning. Quelques années de recherche plus tard, l'algorithme AlphaGo de deepMind (entreprise britannique spécialisée dans l'intelligence artificielle) réussit à battre l'un des meilleurs joueurs de Go au monde, Lee Sedol.

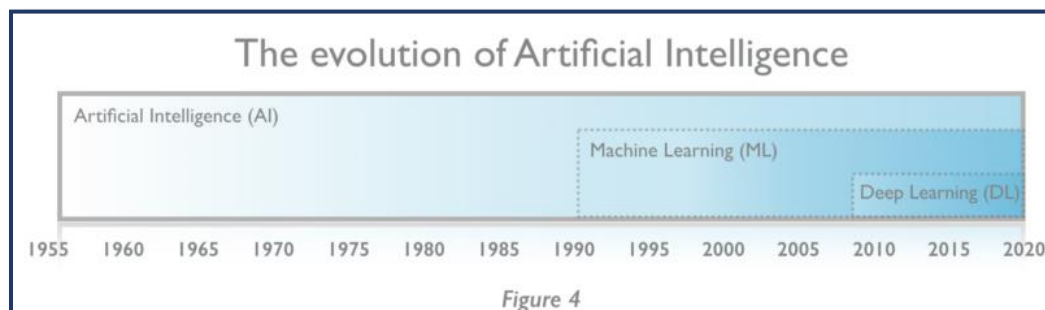


Figure 9 - Evolution vers le deep learning

Le deep learning se base sur un réseau de neurones, comme peut le faire le machine learning, mais celui-ci est dit profond. C'est-à-dire que le réseau est plus complexe et permet de résoudre un plus grand nombre de problématiques.

b- Le réseau de neurones artificiels

L'idée du réseau de neurones artificiels n'est pas neuve, elle remonte à la fin des années 50. Le principe est de s'inspirer de l'architecture du cerveau humain pour y implémenter un algorithme de deep learning. Pour comprendre le fonctionnement d'un réseau de neurones il faut dans un premier temps expliquer ce qu'est un neurone artificiel.

Le neurone artificiel se base sur le neurone biologique présent dans le cerveau humain, celui-ci possède une unique sortie, l'axone qui envoie ou non un signal vers un autre neurone. En entrée, il possède des dendrites qui reçoivent ou non un signal électrique. Et au milieu de ça on trouve le neurone.

Le neurone artificiel mime en quelque sorte ce comportement par une fonction mathématique. Le neurone possède une ou plusieurs entrées étant des valeurs, ces valeurs sont affectées à des poids et additionnées entre elles. Cette somme est ensuite soumise à une fonction d'activation qui va décider d'envoyer ou non un signal en sortie.

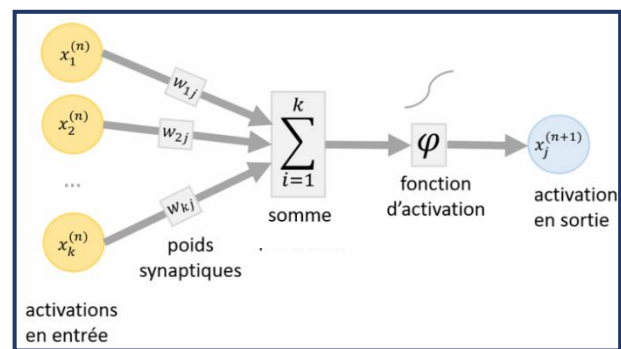


Figure 10 - Schéma d'un neurone artificiel

Le réseau de neurones est par définition une association de plusieurs neurones, allant de quelques dizaines à plusieurs millions pour certains. Ce réseau comprend une couche de neurones d'entrées (input) prenant les valeurs du système, une couche de sorties (output) qui correspond au résultat que l'on veut obtenir et entre les deux il y a une succession de couches dont chacune prend ses entrées sur les sorties de la précédente. C'est ce qu'on appelle un réseau de neurones « fully connected ».

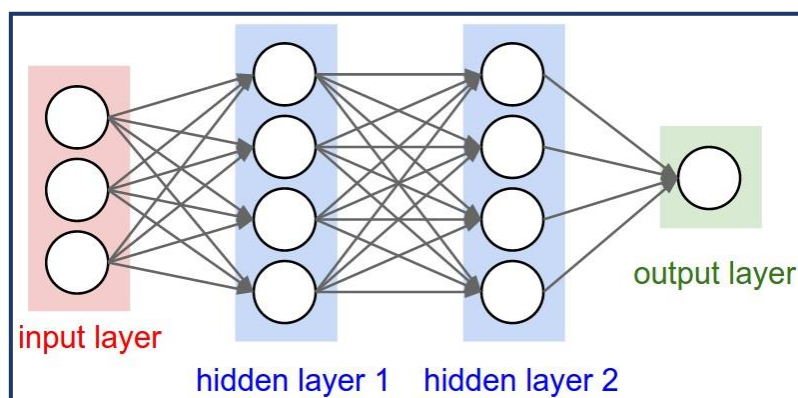


Figure 11 – schéma d'un réseau de neurone « fully connected »

Il existe d'autres types de réseaux dont le plus connu et utilisé est le réseau convolutif. Il est très utile pour faire de la reconnaissance d'image mais possède une structure différente du réseau de neurones « fully connected ». Le fonctionnement d'un réseau de neurones se divise en deux parties : la phase d'apprentissage et la phase de prédiction.

L'apprentissage peut se faire par une approche supervisée, cela signifie qu'on entraîne le réseau en lui transmettant les valeurs ainsi que les résultats que l'on attend. Par exemple pour un réseau convolutif, on insère en entrée une image d'un chien tout en lui indiquant le résultat attendu. Le réseau va comparer la réponse attendue avec la prédiction pour indiquer un taux de précision. Plus on va transmettre d'images, plus le système sera précis. A contrario, l'approche non-supervisée ne transmettra pas les réponses au réseau ce qui veut dire que celui-ci devra de lui-même trouver des relations entre les données. Par exemple, si on donne une image de voiture en entrée, ce système pourra reconnaître les roues, les portes, les surfaces vitrées ... Ce type de modèle demande une grande banque de données d'images et une période d'entraînement assez longue.

Cette période qui consiste à faire passer les données dans le réseau s'appelle la « feed forward propagation ». Mais durant l'apprentissage, le but du réseau est d'optimiser les poids des connexions entre les neurones afin de réduire le coût et augmenter la précision. Les poids sont comme des boutons rotatifs qu'il faut réussir à calibrer pour avoir le meilleur résultat. Pour ce faire, le réseau procède à une « back propagation », cela veut dire que le système va actualiser les poids en partant de la couche de sortie jusqu'à la couche d'entrée. Cela se fait grâce à des calculs plus ou moins complexes invisibles dans le code car effectués par la bibliothèque *tensorflow*.

La phase de prédiction consiste à transmettre des données au réseau afin qu'en sortie, il nous prédise un résultat à partir de l'apprentissage effectué auparavant.

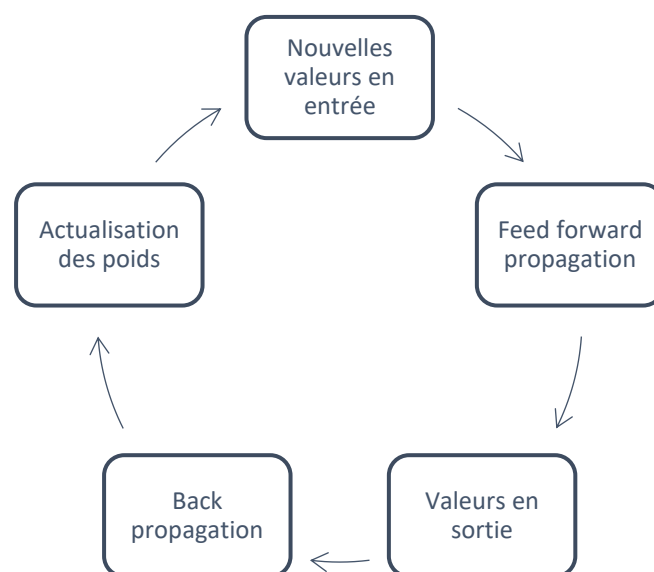


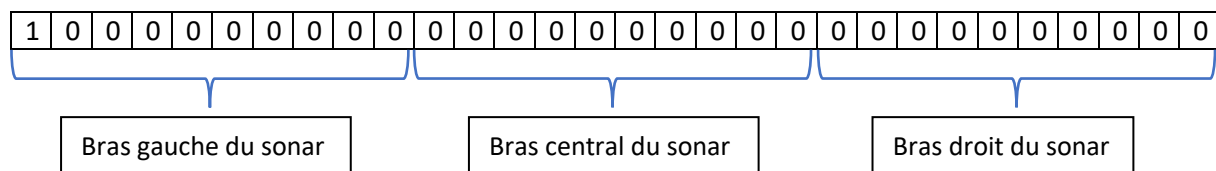
Figure 12 - Cycle de la phase d'apprentissage

c- Mon modèle de réseau de neurones

Le modèle que j'ai choisi pour mon réseau de neurones s'inspire fortement du projet de recherche DeepTraffic mené par des chercheurs de MIT. Celui-ci utilise un réseau de neurones pour qu'une voiture virtuelle, située sur une ligne droite à plusieurs files, avance le plus vite possible tout en évitant les autres voitures présentes sur les voies. Le réseau de neurones prend en entrée les valeurs d'un sonar, comme nous, et renvoie une décision qui ne peut être que rester dans la file, aller dans la file de droite ou aller dans la file de gauche. J'ai adapté ces valeurs de sortie pour qu'elles deviennent une rotation à droite, à gauche ou ne rien faire.

Le réseau comprend une couche d'entrée composée de 30 neurones, trois couches cachées composées de 50 neurones chacune et 3 neurones sur la couche de sortie, chaque neurone correspondant à une direction (droite, gauche, droit devant). C'est un réseau « fully connected » car c'est le plus simple et le plus commun. Tous les poids sont initialisés à zéro à la création du réseau de neurone.

Le couche d'entrée possède 30 neurones car elle correspond au nombre de points du sonar, 10 sur chaque bras, 3 bras donc 30 points. A partir de ces 30 valeurs, on veut obtenir à la sortie du réseau de neurones une direction, mais comment faire passer ces valeurs dans le réseau ? On va convertir le sonar en un vecteur de 1 ligne, 30 colonnes composées de 1 et de 0. Les cases ayant pour valeur 1 correspondent à une collision.



L'exemple ci-dessous représente le vecteur que l'on va passer en entrée du réseau de neurones. Ce vecteur indique que le sonar est entré en collision avec un obstacle situé sur la gauche. On espère donc que le réseau de neurones nous indiquera de tourner à droite, représenté par le vecteur en sortie suivant :

0	0	1
---	---	---

Il ne peut y avoir qu'un seul 1 parmi les 10 cases de chaque bras, on détecte toujours la collision la plus proche, peu importe celle d'après. Cela fait donc 11^3 combinaisons possibles. Voyons dans la partie suivante comment j'ai implémenté et entraîné ce modèle en me basant sur ces 1 331 combinaisons.

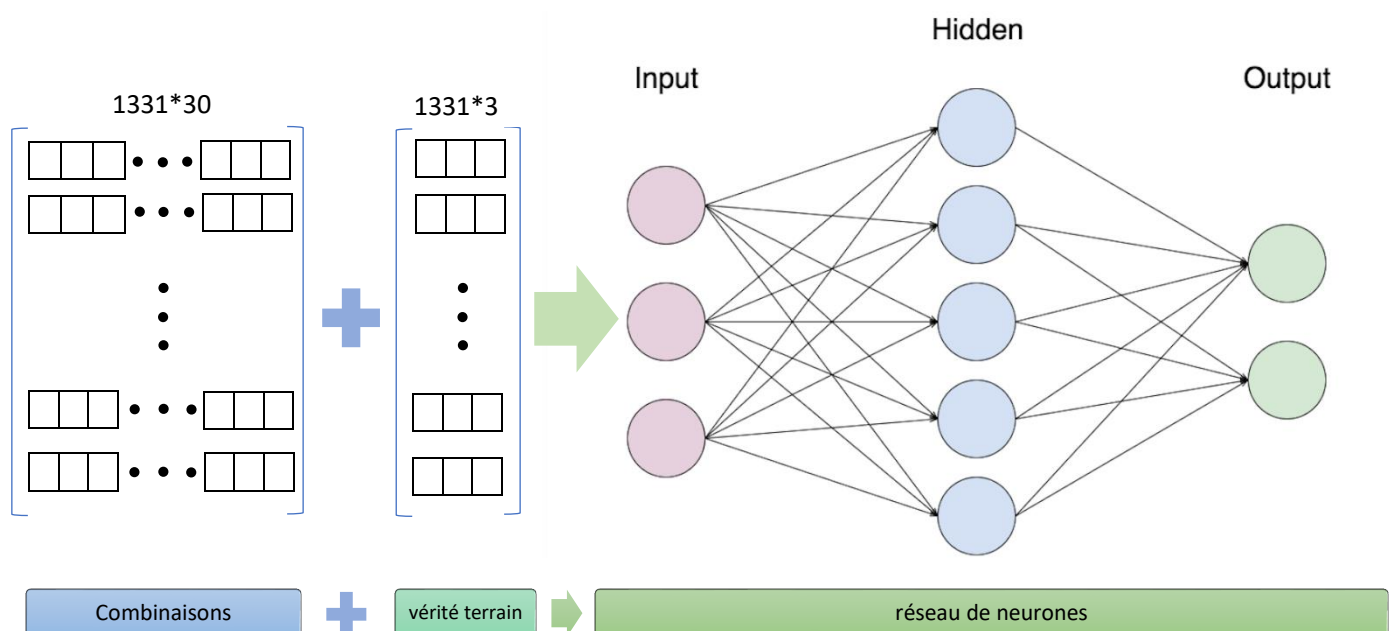
d- L'implémentation du réseau sur le drone

La dernière étape était donc l'implémentation de ce réseau de neurones sur le drone simulé. Pour ce faire j'ai créé une classe « `neural_network` » qui contient 3 fonctions permettant la création du réseau, son entraînement et une dernière fonction de prédiction.

La fonction de création du réseau permet de l'initialisation du nombre de couches, leur nombre de neurones ainsi que les poids.

Pour entraîner le réseau, on fait appel à la fonction « `train_network` ». Mais pour cela, il faut auparavant créer un tableau avec toute les combinaisons ainsi que la vérité terrain, mais qu'est-ce que la vérité terrain ? C'est un tableau comprenant les résultats voulus en fonction des combinaisons. Opérant avec une approche supervisée, notre réseau a donc besoin de la vérité terrain. On génère donc les combinaisons et la vérité terrain avec deux scripts et on obtient donc un tableau de combinaisons (1331×31) et un tableau de la vérité terrain (1331×3).

Le but est ensuite de prendre au hasard un certain nombre d'échantillons dans ces tableaux pour ensuite les passer un par un dans le réseau.



On répète ce processus de tirage aléatoire plusieurs fois pour augmenter la précision. Ainsi, en entrainer le réseau 20 fois avec environ 900 échantillons par itération on passe d'une précision de 50% pour la première itération à 98% lors de la 20^{ème}.

On possède maintenant un réseau entraîné et prêt à recevoir les données du sonar pendant la simulation. Le principe est que l'on va, à chaque « tick » de l'horloge *Pygame*, convertir le sonar en vecteur et l'envoyer à la fonction « prediction » de la classe « *neural_network* ». Cette fonction renvoie le vecteur indiquant la direction à prendre.

1	0	0	→	Rotation de 6° à bâbord
0	1	0	→	Pas de rotation
0	0	1	→	Rotation de 6° à tribord

Cette fonction s'ajoute aux fonctions de mouvement et de vitesse déjà existantes et permet au drone de suivre un trajet tout en évitant les obstacles qui apparaissent devant lui.

Conclusion

Ces six semaines de stages m'auront dans un premier temps permis d'apprendre un nouveau langage, le python, et d'utiliser plusieurs librairies utiles à la réalisation d'interfaces graphiques. Les IHM en python n'ont donc plus aucun secret pour moi désormais.

Mais la partie la plus intéressante fut définitivement celle sur l'intelligence artificielle, et plus particulièrement le deep learning. Elle m'a permis d'enlever les a priori que j'avais sur l'IA, domaine qui m'échappait un peu et me paraissait être assez difficile à comprendre.

J'ai pu découvrir les réseaux de neurones, une technologie que je ne connaissais pas. C'est une approche de deep learning à la fois très intéressante mais aussi difficile à cerner. J'ai donc passé beaucoup de temps à regarder des workshop réalisés par Google et des conférences présentées par Yann LeCun. Cela m'aura permis d'enrichir mes connaissances dans le domaine du deep learning mais aussi d'augmenter mon intérêt pour celui-ci.

Je ne pense pas effectuer une carrière dans le domaine de l'intelligence artificielle car mon projet professionnel est, depuis quelques temps déjà, de devenir professeur de math-info dans un lycée. Néanmoins je continuerai à garder un œil sur l'avancement de la recherche en matière de réseaux de neurones car elle est très prometteuse. Les puissances de calculs dans les années à venir permettront peut-être au réseau de neurones de s'approcher de plus en plus de l'architecture du cerveau humain et qui sait, permettre l'éveil d'une intelligence artificielle.

Annexes

Les codes, le tutoriel ainsi que le diagramme de classe complet sont disponibles sur le github :

Adresse GitHub à mettre

Voici le lien donné par Mr Ménard qui m'a inspiré pour la création du sonar :

Matt Harvey. *Using reinforcement learning in Python to teach a virtual car to avoid obstacles.*

<https://medium.com/@harvitronix/using-reinforcement-learning-in-python-to-teach-a-virtual-car-to-avoid-obstacles-6e782cc7d4c6>

Le projet communautaire DeepTraffic utilisant les réseaux de neurones qui présente des similarités avec notre projet de simulateur : <http://selfdrivingcars.mit.edu/deeptraffic/>

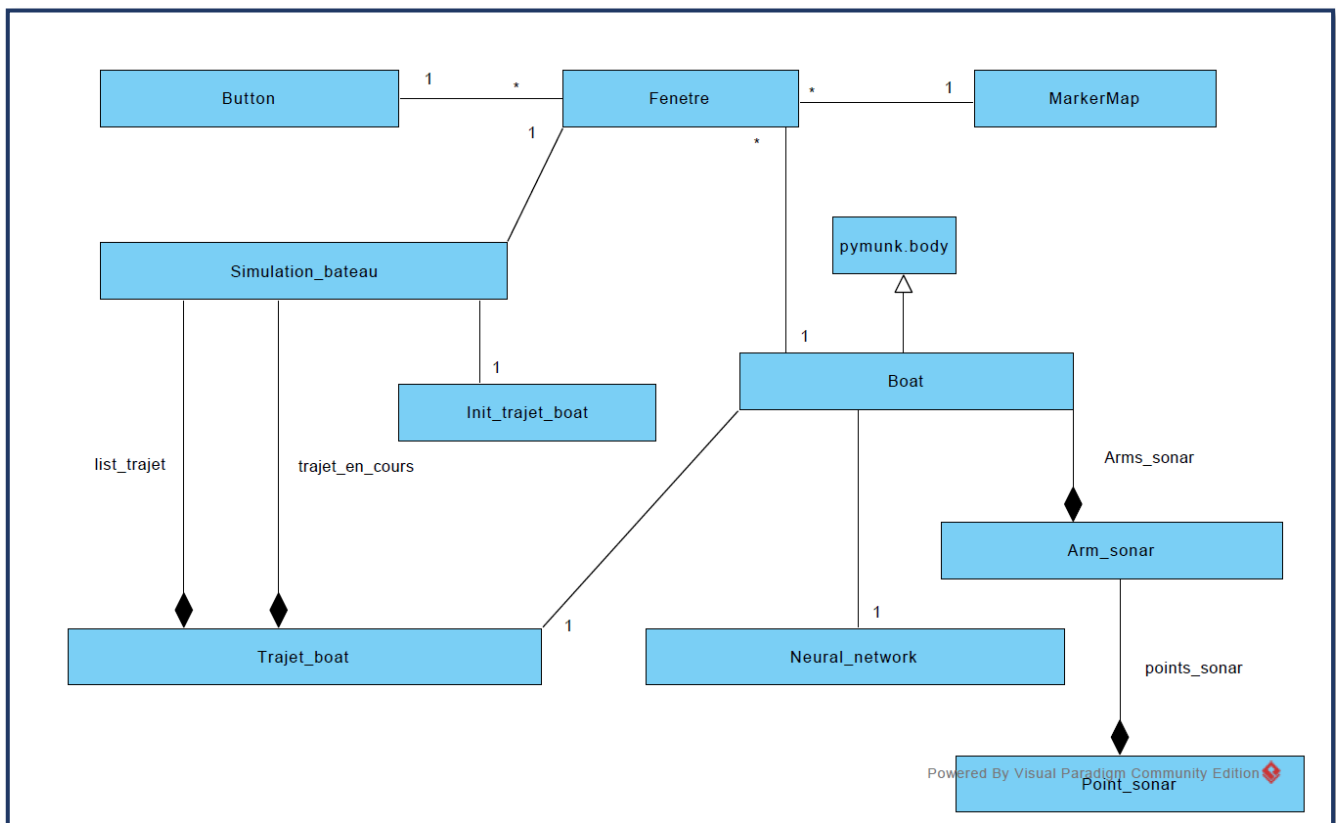


Figure 13 - Diagramme de classe simplifié du simulateur

Le projet DeepTesla a pour objectif d'éviter les obstacles sur la route, le projet du L3i est relativement semblable mais se passe sur l'eau : <http://selfdrivingcars.mit.edu/deeptesla/>

Bibliographie / sitographie

Ian Goodfellow, Yoshua Bengio et Aaron Courville. *Deep Learning*. The MIT Press, 2016. (Adaptive Computation and Machine Learning series)

David Louapre. *Le deep learning – Science étonnante #27*. Vidéo Youtube
<https://www.youtube.com/watch?v=trWrEWfhTVg>

Pirmin Lemberger et Manuel Alves. *Le Deep learning pas à pas*. Site d'entreprise.
<https://www.technologies-ebusiness.com/enjeux-et-tendances/le-deep-learning-pas-a-pas>

Udacity. *Lesson 3 : Deep Neural Networks*. Cours en ligne.
<https://classroom.udacity.com/courses/ud70>

Yann LeCun. *L'apprentissage profond : une révolution en intelligence artificielle*. Conférence.
<http://www.college-de-france.fr/site/yann-lecun/inaugural-lecture-2016-02-04-18h00.htm>

Faizan Shaikh. *Simple Beginner's guide to reinforcement Learning & its implementation*. Site communautaire. <https://www.analyticsvidhya.com/blog/2017/01/introduction-to-reinforcement-learning-implementation>

Yann LuCun. *Deep learning – Yann LeCun, à l'USI*. Vidéo Youtube d'une conférence.
https://www.youtube.com/watch?v=RgUcQceqC_Y

Steven Miller. *Mind : How to build a Neural Network*. Site personnel.
<https://stevenmiller888.github.io/mind-how-to-build-a-neural-network>

Faizan Shaikh. *An Introduction to implementing Neural Network using tensorflow*. Site communautaire. <https://www.analyticsvidhya.com/blog/2016/10/an-introduction-to-implementing-neural-networks-using-tensorflow>

Déborah Mesquita. *Big Picture Machine Learning: Classifying Text with Neural Networks and TensorFlow*. Site communautaire. <https://medium.freecodecamp.com/big-picture-machine-learning-classifying-text-with-neural-networks-and-tensorflow-d94036ac2274>

Julien Lausson. *TensorFlow : DeepMind embrasse l'IA de Google pour ses futurs projets*. Journal d'information en ligne. <http://www.numerama.com/sciences/167824-tensorflow-deepmind-embrasse-lia-de-google-pour-ses-futurs-projets.html>

Morgane Tual. *Comment le « deep learning » révolutionne l'intelligence artificielle*. Journal d'information en ligne. http://www.lemonde.fr/pixels/article/2015/07/24/comment-le-deep-learning-revolutionne-l-intelligence-artificielle_4695929_4408996.html