

## Devoir 3 - Reconnaissance de plan dans un domaine de navigation sur une grille

### Objectifs

- Se familiariser avec diverses approches de la reconnaissance de plan.
- Tester des algorithmes de reconnaissance de plan dans un environnement simple et intuitif de navigation dans une carte.
- Comprendre et implémenter une méthode de reconnaissance de plan symbolique.
- Approcher la reconnaissance d'activités via l'apprentissage-machine.
- Utiliser diverses approches connexionnistes et appréhender l'impact des paramètres sur la reconnaissance d'activités.
- Comparer la reconnaissance de plan symbolique et par apprentissage.

### Consignes.

- Le devoir est le même pour les étudiants de IFT608 et IFT702.
- Le devoir 3 se fait en équipe d'au plus 4 étudiants.
- La remise se fait dans Moodle sous la forme de deux archives (*d3-etape1.zip* et *d3-etape2.zip*).
- Les remises doivent respecter scrupuleusement les consignes données dans l'énoncé. Un non-respect des consignes qui entraîne une complication dans la correction sera pénalisé en fonction de l'écart des consignes.

### Grille d'évaluation

Étape 1. Reconnaissance de plan symbolique	/30
Étape 2. Apprendre à reconnaître des plans	/30
<b>Total</b>	<b>/60</b>
<b>Bonus - Étape 2. Étude du modèle <i>CNNMultimaps</i></b>	<b>/5</b>

## Présentation

L'objectif de ce travail est de se familiariser avec des méthodes de reconnaissance de plan et de les tester dans un environnement simple et intuitif de navigation dans une carte. Par exemple, vous soupçonnez un ennemi (point bleu) de vouloir attaquer une de vos bases (points orange), la reconnaissance de plan essaiera d'identifier quelle base il veut attaquer au fur et à mesure que vous observez sa progression, (Figure 1).

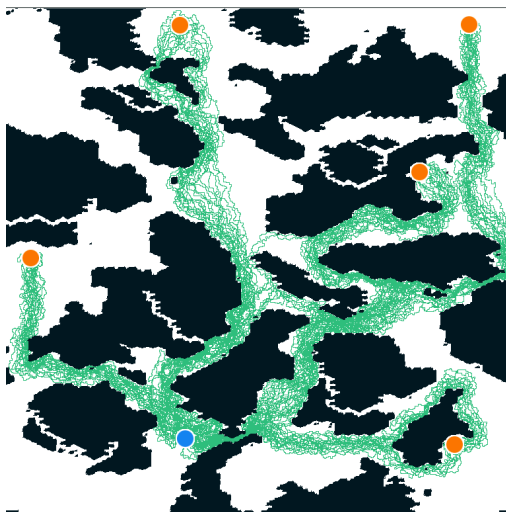


Figure 1 : Exemple de chemins observés dans une carte de StarCraft. La position initiale est le point bleu et les buts possibles sont les points orange. Un chemin est représenté en vert. Il y a 20 chemins observés pour chaque but.

Dans un premier temps, vous explorerez la reconnaissance de plan symbolique. La reconnaissance de plan s'appuie sur l'hypothèse de la rationalité de l'agent observé. La première étape consistera ainsi à comprendre et implémenter la méthode de Masters et Sardina publiée en 2017<sup>1</sup>.

Peta Masters, and Sebastian Sardina, Cost-Based Goal Recognition for Path-Planning Proc. of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017), S. Das, E. Durfee, K. Larson, M. Winikoff (eds.), May 8–12, 2017, São Paulo, Brazil, pp. 750-758

Dans un second temps, vous explorerez l'application de l'apprentissage à la reconnaissance de plan. La reconnaissance de plan est alors reformulée comme un problème d'apprentissage. Vous aurez alors à suivre la démarche et reproduire les expérimentations décrites dans

Maynard, Mariane, Thibault Duhamel, Froduald Kabanza, Symbolic Inverse-Planning vs Deep-Learning Plan Recognition, The AAAI 2019 Workshop on Plan, Activity, and Intent Recognition, 2019, 8 p

---

<sup>1</sup> L'article est disponible dans Moodle et à cette adresse: <http://www.ifaamas.org/Proceedings/aamas2017/pdfs/p750.pdf>

# 1 Reconnaissance de plan symbolique – Étape 1

L'idée est la suivante : étant donnée une séquence de positions, représentant le trajet d'un agent, on veut déterminer à quel but il est le plus probable qu'il se rende. L'algorithme de Masters et Sardina est un algorithme de planification inverse et est d'ailleurs une adaptation de l'algorithme de Ramirez et Geffner (2010) dans le cas particulier de la navigation<sup>2</sup>. Il se base sur un principe élémentaire de rationalité : on suppose qu'un agent essaye toujours d'agir d'une manière optimale selon les informations qu'il possède. Cela veut dire que, dans le domaine de navigation où l'on se place actuellement, un agent va toujours essayer de se déplacer vers son but avec un coût optimal (ou du moins le plus petit possible). Ainsi, en se basant sur le coût d'un chemin observé, il est possible de calculer les probabilités qu'il se rende à chacun des buts possibles. L'algorithme 1 présente la méthode à suivre.

---

**Algorithm 1** Calcul des probabilités

---

Let  $O = o_1, o_2, \dots, o_n$  be an observation sequence  
Let  $s$  stand for the start position  
Let  $G$  stand for the set of possible goals  
Let  $pos$  stand for the last observed position ( $pos \leftarrow o_n$ )  
**for** each goal  $g$  in  $G$  **do**  
     $cost(s, g) \leftarrow$  cost of an optimal path from  $s$  to  $g$   
     $cost(pos, g) \leftarrow$  cost of an optimal path from  $pos$  to  $g$   
     $\Delta(s, pos, g) \leftarrow cost(pos, g) - cost(s, g)$   
     $obsLikelihood[g] \leftarrow \frac{e^{-\Delta(s, pos, g)}}{\dots}$   
**end for**

Normalize the observations likelihood vector into a posterior probability vector that sums up to 1

$$postProbs[g] = \frac{obsLikelihood[g]}{\sum obsLikelihood}, \forall g \in G$$

Select the goal with the higher probability

---

## 1.1 Environnement de travail

Les fichiers à utiliser sont à télécharger ici dans Moodle. Ils vont se trouver dans le répertoire « PlanRecognitionNavigationIFT608-p1 ». Voici les outils à installer afin de pouvoir faire fonctionner le code :

- Python 3.5
- Numpy (version correspondante à Python3.5)
- h5py (version correspondante à Python3.5)

Il n'est pas nécessaire d'installer un environnement virtuel python, mais cela simplifie la gestion des versions. Il est également possible d'utiliser une version spécifique de python dans le terminal, avec la commande python3.5.

---

<sup>2</sup> M. Ramirez and H. Geffner. Probabilistic plan recognition using off-the-shelf classical planners. In Proceedings of the National Conference on Artificial Intelligence (AAAI), pages 1121–1126, 2010.

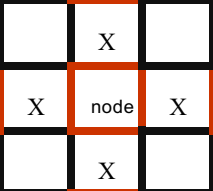
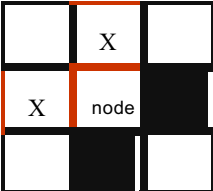
<https://www.dtic.upf.edu/~hgeffner/pr-aaai-2010.pdf>

## 1.2 Tâches à réaliser

La première partie consiste à implémenter la méthode de Masters et Sardina en remplissant le fichier de code *MS.py*. Les ajouts doivent seulement se faire là où la mention *#Add Code Here* est spécifiée.

### 1.2.1 Grid world

Ces modifications sont à effectuer dans la classe *world* du fichier python. Elle est utilisée pour modéliser le problème. Dans la grille, les cases navigables sont des instances de la classe *Node([x,y])* et les obstacles sont *None* (Figure 2). On suppose, en outre, que le coût de chaque action est de 1.

	
Ici, la fonction doit retourner les 4 nœuds voisins dans la grille (marqués d'un X)	Ici, la fonction doit retourner les 2 nœuds voisins qui ne sont pas des obstacles (marqués d'un X)

**Figure 2:** Exemple de comportement attendu pour la fonction *children*.

- Implémentez la fonction *children*. Elle doit retourner les enfants explorables d'un nœud dans la grille. Les seules actions possibles pour l'agent sont les 4 déplacements cardinaux (Nord, Sud, Est, Ouest), chacun de coût unitaire. De plus, l'agent ne pas aller sur une case s'il y a un obstacle et il ne faut pas non plus que la case soit en dehors de la grille.
- Implémentez la fonction *heuristic* avec l'expression de votre choix. Elle doit être admissible.
- Implémentez la fonction *planner*. Elle doit retourner une liste de nœuds adjacents formant un chemin valide d'un nœud de départ à un nœud d'arrivée. Il convient d'utiliser l'algorithme de votre choix, mais il faut absolument qu'il retourne un chemin de coût **optimal**. A\* est suggéré.

### 1.2.2 Implémentation de Masters et Sardina

Maintenant que les outils de modélisation ont été mis en place, cette partie consiste à implémenter la méthode en elle-même.

- Implémentez la méthode de reconnaissance de plan de Masters et Sardina dans la fonction *predictMastersSardina*. Cette fonction doit retourner la distribution des probabilités à posteriori telle que définie par l'algorithme 1.
- Implémentez la fonction *accuracy*. Elle correspond à la précision de la méthode. Elle permet d'évaluer l'efficacité des prédictions et est simplement égale au rapport du nombre de prédictions correctes sur le nombre de prédictions totales (multipliée par 100 pour un pourcentage).

$$accuracy = \frac{\#correct\ predictions}{\#predictions} \times 100$$

Une prédiction est correcte si le but ayant la plus grande probabilité est le vrai but étiqueté. Attention au cas d'égalité des probabilités maximales. Par exemple, si les probabilités calculées sont  $P(A)=0.4$ ,  $P(B)=0.4$  et  $P(C)=0.2$  et que le vrai but étiqueté est B, alors on considère que la prédiction est correcte, même si en pratique il faudrait choisir au hasard.

### 1.3 Critères d'évaluation

L'étape 1 est noté sur **30 points**. Les points seront distribués de la façon suivante :

- Implémentation de *children* : **3 points**
- Implémentation de *heuristic* : **3 points**
- Implémentation de *planner* : **7 points**
- Implémentation de *predictMastersSardina* : **7 points**
- Implémentation de *accuracy* : **3 points**
- Exécution : **7 points**

L'évaluation de chacun de ces points se fera selon les critères suivants :

- Exécution sans erreur et fonctionnelle;
- Clarté et qualité du code et de la documentation;
- Explication et justification des choix d'implémentation;

### 1.4 Livrables

Vous devez rendre les fichiers suivants :

- MS.py
- Un court rapport ou une vidéo présentant l'exécution de votre programme pour un exemple. Vous pouvez choisir un exemple qui sera représentatif. On doit voir clairement l'évolution du processus de reconnaissance, i.e. au fur et à mesure que l'agent progresse, quelles sont les valeurs des fonctions de coût, quelles sont les prédictions, quelle est la précision (*accuracy*) de chaque prédiction.
- Un fichier readme.md qui contient les noms et matricules des étudiants, puis décrit brièvement pour chaque question vos choix d'implémentations. Voici comment il doit être organisé :

```
Matricule, Nom Prénom
Matricule, Nom Prénom
Matricule, Nom Prénom

#      children
[Description...]

#      heuristic
[Description...]

#      planner
[Description...]

#      predictMastersSardina
[Description...]

#      accuracy
Description...]
```

## 2 Apprendre la reconnaissance de plan – Étape 2

Il existe plusieurs manières d’aborder le problème de reconnaissance de plan : par inférence probabiliste, par planification inverse (*plan recognition as planning*), et par analyse grammaticale (*plan recognition as parsing*). Si vous êtes intéressés à en savoir plus, vous êtes invités à consulter les diapositives du tutoriel sur la reconnaissance de plan qui a eu lieu à l’occasion de AAAI 2019<sup>3</sup>.

Au workshop de PAIR 2019 (Plan, Activity, and Intent Recognition) à AAAI 2019, une autre façon d’aborder le problème a été présentée par Maynard et al. 2019<sup>4</sup> : par apprentissage automatique (*plan recognition as learning*). En effet, un problème de reconnaissance de plan peut être vu comme un ensemble de paires d’exemples  $\langle o_\pi, g \rangle$  où  $o_\pi$  est une séquence d’observations du comportement de l’agent et où  $g$  est le but qu’il poursuit. Inférer le but devient donc un problème de classification qui peut être résolu par apprentissage sur l’ensemble de données.

Pour cette partie, vous devrez reproduire les architectures présentées et résumées dans l’article de Maynard et al. 2019 pour le domaine de navigation. En ayant à la base l’ensemble de données d’entraînement, reproduisez les architectures et les conditions d’expérimentation de l’article, entraînez vos réseaux, testez et comparez toutes les méthodes entre elles, y compris celle de Masters et Sardina que vous avez implémentée à la partie 1. Obtenez-vous des résultats semblables à ceux présentés?<sup>5</sup> Discutez et tentez d’expliquer ce que vous obtenez.

### 2.1 Environnement de travail

Les fichiers à utiliser sont à télécharger ici dans Moodle. Ils vont se trouver dans le répertoire « PlanRecognitionNavigationIFT608-p2 ». Voici les outils à installer afin de pouvoir faire fonctionner le code :

- Python 3.5
- Numpy (version correspondante à Python3.5)
- h5py (version correspondante à Python3.5)
- Keras
- TensorFlow >= 1.10

Il n’est pas nécessaire d’installer un environnement virtuel python, mais cela simplifie la gestion des versions. Il est également possible d’utiliser une version spécifique de python dans un terminal (bash), avec la commande `python3.5`.

Il n’est pas non plus nécessaire d’installer tensorflow-gpu, ni tous les programmes nécessaires pour entraîner des réseaux de neurones sur GPU comme CUDA et CuDNN. Si vous souhaitez tout de même le faire, vous êtes responsables de faire vous-mêmes les configurations nécessaires sans aide du correcteur, et il faudra que votre code s’exécute sans erreur avec la configuration minimale ci-haut.

---

<sup>3</sup> Vous trouverez le fichier dans Moodle ou à l’URL : [http://www.planrec.org/Tutorial/Resources\\_files/pair-tutorial.pdf](http://www.planrec.org/Tutorial/Resources_files/pair-tutorial.pdf).

<sup>4</sup> L’article se trouve dans Moodle.

<sup>5</sup> Étant donné que les problèmes ont été simplifiés, il ne faut pas forcément obtenir les mêmes résultats que ceux décrits dans l’article.

## 2.2 Tâches à réaliser

Cette partie consiste à faire de l'apprentissage automatique pour résoudre des problèmes de reconnaissance de plan dans le domaine de navigation. Vous devrez implémenter deux des quatre architectures présentées dans l'article [Maynard et al. 2019]<sup>6</sup> avec Keras (résumées dans le fichier « UdeS-Jan-2019.pdf »). Comme dans la partie 1, une base de code vous est fournie pour traiter les données sur lesquelles vous allez entraîner et tester les modèles. Pour l'entraînement, utilisez l'algorithme d'optimisation Adam avec ses paramètres par défaut ( $\beta_1 = 0.9$  et  $\beta_2 = 0.999$ ) et un taux d'apprentissage de 0.001. Quatre modèles sont étudiés dans l'article: fully connected, LSTM, CNN et CNNMultimaps. Dans cette étape, vous devez

1. Comprendre comment reconnaître des plans à l'aide de l'apprentissage automatique
2. Choisir 2 modèles parmi les 4 modèles étudiés dans l'article et les comparer avec l'approche de Masters et Sardina (qui représente l'état de l'art au niveau de la planification symbolique).
3. Comprendre les paramètres de configuration des 2 modèles que vous avez choisis et leur impact sur l'apprentissage réalisé.

Un script d'entraînement et de test vous sera fourni sous la forme de fichiers « *TypeDeRéseauRun.py* »<sup>7</sup>. Le script de test consiste à faire les expérimentations, mais avec des observations partielles du comportement de l'agent, allant de 25% à 100% du chemin complet. Une fois l'implémentation et l'entraînement complétés, testez vos architectures et prenez les valeurs retournées lors du test pour les placer dans un graphe similaire à celui de la Figure 3 de l'article. Des fichiers de solutions vous sont fournis pour chacun des types de réseau à utiliser (PlanRecognitionNavigationIFT608-p2-Solution.zip). Vous devez comprendre ces programmes.

Dans les fichiers solutions qui vous sont donnés, les paramètres de configuration des réseaux, des apprentissages et des tests sont déjà établis. Pour la plupart des paramètres de configuration, les valeurs par défaut sont utilisées. Par exemple pour LSTM, la valeur par défaut « sigmoid » est utilisé pour le paramètre « recurrent\_activation ». Dans certains cas, des valeurs spécifiques sont choisies. Par exemple pour LSTM, la valeur « 0.1 » est utilisé pour le paramètre « recurrent\_dropout ». Pour chacun des deux modèles que vous avez choisis, vous devez prendre au moins deux paramètres et les faire varier pour en montrer l'impact (ou l'absence d'impact) sur la reconnaissance de plan.

Les sections qui suivent présentent les configurations et les paramètres des 4 modèles que vous pouvez étudier. Vous devez en choisir deux parmi les quatre.

### 2.2.1 Architecture fully connected

Implémentez<sup>8</sup> l'architecture *fully connected* (FC) pour le domaine de navigation dans le fichier « *fullyConnected.py* ». L'entrée sera composée de 10 observations sous forme de coordonnées  $(x, y)$  et la sortie, de 5 valeurs entre 0 et 1 représentant la probabilité *a posteriori* de chacun des buts. Il y aura toujours 5 buts pour chacun des problèmes sur lesquels vous allez entraîner votre réseau. Voici l'architecture en détails :

1. Couche / sortie (dense) de 5 unités avec activation *softmax*.

Utilisez l'entropie croisée pour catégories multiples (*categorical\_crossentropy*) comme fonction de perte.

---

<sup>6</sup> Maynard, Mariane, Thibault Duhamel, Froduald Kabanza, Symbolic Inverse-Planning vs Deep-Learning Plan Recognition, The AAAI 2019 Workshop on Plan, Activity, and Intent Recognition, 2019, 8 p.

<http://www.planrec.org/PAIR/PAIR19/Resources.html>, cliquez sur "ACCEPTED PAPERS" dans le menu de gauche pour télécharger les articles. C'est l'article "PAIR-19\_paper\_25.pdf". L'article est aussi disponible dans Moodle.

<sup>7</sup> Plus précisément, CNNRun.py, CNNMultimapsRun.py, fullyConnectedRun.py et LSTMRun.py

<sup>8</sup> Même si une implémentation vous est donnée, essayez d'abord d'esquisser la solution sans regarder la réponse.

Vous aurez peut-être besoin de reformater la dimension de l'entrée pour être praticable par les couches denses (*fully connected*).

Pour tester votre architecture, utilisez simplement le fichier python « `fullyConnectedRun.py` ».

Dans un second temps, prenez au moins deux paramètres et faites-les varier pour en montrer l'impact (ou l'absence d'impact) sur la reconnaissance de plan. Essayez d'expliquer cet impact.

### 2.2.2 Architecture LSTM

Implémentez le réseau *Long Short-Term Memory* (LSTM) de l'article [Maynard et al. 2019] pour le domaine de navigation défini dans le fichier « `LSTM.py` ». L'entrée est la même que pour la première architecture (composée de 10 observations sous forme de coordonnées  $(x, y)$ ) de même que pour la sortie (composée de 5 valeurs entre 0 et 1 représentant la probabilité *a posteriori* de chacun des buts). Voici l'architecture en détails :

1. Couche LSTM de 100 unités, activation ReLU, et 'dropout' (y compris 'dropout récurrent') avec probabilité de 'drop' à 0.1;
2. Couche de sortie (dense) de 5 unités avec activation *softmax*.

Utilisez l'entropie croisée pour catégories multiples (*categorical crossentropy*) comme fonction de perte. Notez que c'est un problème 'sequence-to-one' (vs 'sequence-to-sequence'). Votre réseau devrait retourner une seule valeur à la fin de la séquence.

Pour tester votre architecture, utilisez simplement le fichier python « `LSTMRun.py` ».

Dans un second temps, prenez au moins deux paramètres et faites-les varier pour en montrer l'impact (ou l'absence d'impact) sur la reconnaissance de plan. Essayez d'expliquer cet impact.

### 2.2.3 Architecture à convolutions

Implémentez le réseau à convolutions (CNN) de l'article [Maynard et al. 2019] pour le domaine de navigation dans le fichier « `CNN.py` ». Cette fois-ci, l'entrée est une image 64x64 pixels à 4 canaux correspondant à la carte sur laquelle navigue l'agent. Chaque canal représente un aspect du problème : position de l'état initial, positions des 5 buts potentiels, les positions visitées (c'est-à-dire du chemin observé), et les cases praticables qui ne correspondent à aucune des positions ci-haut. Chaque case d'un canal a soit une valeur de 0 ou 1, indiquant si c'est une position initiale, un but potentiel, une position visitée, ou une position praticable, respectivement. Si les valeurs sont toutes à 0 pour tous les canaux d'une case, c'est que c'est un obstacle. La sortie, quant à elle, est du même format que pour les 2 premières architectures, c'est-à-dire 5 valeurs entre 0 et 1 représentant la probabilité *a posteriori* de chacun des buts. Voici l'architecture en détails :

1. Une suite de 6 couches d'architecture identiques (mais avec des paramètres différents) à convolutions composées de 4 filtres de taille 5x5, avec 'zero-padding' en mode 'same' (donc de 2 pour 5x5), et 'strides' de 1, chacune suivie d'une activation Relu et 'dropout' avec probabilité de 'drop' à 0.1;
2. Une couche à convolution avec 1 seul filtre 5x5, 'zero-padding' en mode 'same', 'strides' de 1, activation ReLU et 'dropout' avec probabilité de 'drop' de 0.1;
3. Couche de sortie (dense) de 5 unités avec activation *softmax*. (N'oubliez pas de reformater la dimension de la sortie de la dernière couche à convolution en conséquence).

Utilisez encore l'entropie croisée pour catégories multiples (*categorical crossentropy*) comme fonction de perte.

Pour tester votre architecture, utilisez simplement le fichier python « `CNNRun.py` ».

Dans un second temps, prenez au moins deux paramètres et faites-les varier pour en montrer l'impact (ou l'absence d'impact) sur la reconnaissance de plan. Essayez d'expliquer cet impact.



### 2.2.4 Architecture *fully convolutional* pour faire de la reconnaissance de plan sur plusieurs cartes

Implémentez le réseau à convolutions de l'article [Maynard et al. 2019] pour le domaine de navigation qui permet de faire de la reconnaissance de plans sur plusieurs cartes défini dans le fichier « CNNMultimaps.py ». L'entrée est la même que pour le problème précédent, c'est-à-dire une image 64x64 pixels à 4 canaux correspondant à la carte sur laquelle navigue l'agent. La sortie, cette fois-ci, sera aussi une image 64x64 où chaque pixel/case contient une valeur entre 0 et 1, représentant la probabilité que le but poursuivi par l'agent se trouve à cette position. Voici l'architecture en détails :

1. Une suite de 6 couches d'architecture à convolutions identiques composées de 4 filtres de taille 8x8, avec 'zero-padding' en mode 'same', et 'strides' de 1, chacune suivie d'une activation Relue et 'dropout' avec probabilité de 'drop' à 0.1;
2. 1 couche à convolution avec 1 seul filtre 8x8, 'zero-padding' en mode 'same', 'strides' de 1, 'dropout' avec probabilité de 'drop' de 0.1;
3. Activation *softmax* sur les pixels du 'feature map' généré. (N'oubliez pas de reformer la sortie en conséquence. Softmax s'appliquera au niveau des canaux à moins que vous n'aplatissiez ('flatten') l'image préalablement).

Utilisez encore l'entropie croisée pour catégories multiples (*categorical crossentropy*) comme fonction de perte. Cette-fois, la cible sera un vecteur 'one-hot' de taille 64 \* 64 où la valeur 1 se trouve au 'pixel' correspondant. Si votre sortie a le bon format, vous ne devriez pas rencontrer de problèmes à utiliser cette fonction de perte comme auparavant.

Pour tester votre architecture, utilisez simplement le fichier python « CNNMultimapsRun.py ».

Dans un second temps, prenez au moins deux paramètres et faites-les varier pour en montrer l'impact (ou l'absence d'impact) sur la reconnaissance de plan. Essayez d'expliquer cet impact.

Note : Ce modèle risque d'être un peu plus difficile ou demander un peu plus de travail que les précédentes. Un bonus est accordé si vous le choisissez et que votre solution est fonctionnelle.

## 2.3 Critères d'évaluation

L'étape 2 est noté sur **30 points**. Les points seront distribués de la façon suivante :

- Résolution du problème : code fonctionnel, variation des paramètres, jeux d'essais... **(6 points)**
- Impact des paramètres de configuration sur le comportement du réseau de neurones et sur la reconnaissance de plan pour un modèle **(12 points)**
  - Étude de *fully connected*: 6 points
  - Étude de *LSTM* : 6 points
  - Étude de *CNN* : 6 points
  - Étude de *CNNMultimaps* : 6 points + **bonus (5 points)**
- Comparaison des prédictions entre les deux modèles connexionnistes choisis (étape 2) et l'approche symbolique (étape 1) **(12 points)**

L'évaluation de chacun de ces points se fera selon les critères suivants :

- Exécution sans erreur et fonctionnelle;
- Clarté et qualité du code et de la documentation;
- Justification des choix de variables à faire varier;
- Explication et justification des résultats obtenus;
- Maîtrise des approches utilisées;
- Qualité de la discussion.

## 2.4 Livrables

Vous devez rendre

- Les fichiers suivants que vous avez utilisés pour vos calculs (2 parmi 4):
  - « fullyConnected.py »
  - « LSTM.py »
  - « CNN.py »
  - « CNNMultimaps.py »
- Un rapport (max. 10 pages<sup>9</sup>) dans lequel vous allez présenter
  - Pour chaque type de réseau, les paramètres que vous avez fait varier, pourquoi vous les avez choisis, quelles étaient vos prédictions sur leur impact dans les résultats, quel a été leur impact (ou l'absence d'impact) sur le comportement du réseau de neurones et sur la reconnaissance de plan
  - Les résultats que vous avez obtenus dans un graphique similaire à celui de la Figure 3 de l'article [Maynard et al. 2019]. Discutez les résultats obtenus. Voici le type de questions auxquelles vous pourriez répondre : Est-ce que les résultats obtenus sont les mêmes que ceux de l'article? S'ils sont différents, pourquoi pensez-vous qu'ils le sont? Quelles sont les principales différences? etc.
- Un fichier « readme.md » qui contient les noms et matricules des étudiants, puis décrit brièvement pour chaque question l'architecture (qui devrait être telle que décrite dans l'énoncé). Voici comment il doit être organisé :

```
Matricule1, Prénom1 Nom1
Matricule2, Prénom2 Nom2
Matricule3, Prénom3 Nom3
```

```
# fullyConnected
[Description...]
```

```
# LSTM
[Description...]
```

```
# CNN
[Description...]
```

---

<sup>9</sup> C'est un maximum, vous n'êtes pas obligés de produire un document de 10 pages.