SIMON-FINE Thibaut
SÉNÉCHAL Sylvain

Projet COCOMA

RoboCup Rescue - Qlearning

I. Introduction

L'environnement de RoboCup Rescue propose une simulation complexe d'événements de types "désastres naturels". Il nous amène à réfléchir et élaborer diverses techniques permettant de réagir à de telles situations de façon optimale. Les problèmes de gestion de crises étants considérés comme très difficiles à résoudre lorsqu'ils sont modélisés correctement, puisque de nombreux paramètres rentrent en compte : **multiples** agents, agents aux **rôles différents**, **informations incomplètes** et **communication imparfaite**.

II. Aspects techniques

La première étape avant de commencer le travail sur le projet fut de rechercher et d'obtenir une compréhension du fonctionnement de la plateforme, ainsi que de son architecture de code. Une fois cela effectué nous avons ensuite décidé des support que nous allions utiliser pour réaliser le projet.

Le projet est dans son ensemble disponible sur un répertoire github :
https://github.com/ThibautSF/COCOMA_RobotRescue

A. Structure du projet

Le projet contient les dossiers suivants :

- rcrs-server : serveur de simulation de RoboCup Rescue, clone du dépôt https://github.com/roborescue/rcrs-server
- rcrs-adf-stss : l'Agent Development Framework[1] de RoboCup Rescue, dans lequel nos modifications ont été effectuées
- maps : les cartes personnalisées pour ce projet (soit uniquement les incendies)

[1] Lien vers ADF : https://github.com/roborescue/rcrs-adf-sample

SIMON-FINE Thibaut
SÉNÉCHAL Sylvain

Le dossier ‘rcrs-adf-stss’ contient diverses modifications par rapport au répertoire original, le package ‘stss.qlearningproject’ y est ajouté dans les sources java. Ce dernier reprend la structure données par la documentation de RoboCup :

- stss.qlearningproject.centralized : pour le code des agents centralisés, non utilisé dans le cadre du projet
- stss.qlearningproject.extaction : pour le code des actions combinées, ajout du classes implémentant l’apprentissage
- stss.qlearningproject.module
    - algorithm
    - complex
    - qlearning : package ajouté dans le cadre du projet, toutes les classes du package servent à l’implémentation de l’algorithme du Qlearning et ses utilitaires associés (diverses stratégies d’exploration, initialisation ainsi que fonctions d’import et export)

B. QLearning

Le QLearning a été programmé dans la classe éponyme, on l'instancie en lui passant un nombre d’états et d’actions qui génèrera alors une matrice de taille adéquat à 0, puis il suffit d’appeler la méthode ‘update’ pour mettre à jour la Qtable en fonction de la reward obtenue après une action.

Concernant l’obtention par le Qlearning d’une action à partir d’un état donné, nous avons choisi d’implémenter plusieures stratégies d’explorations (toutes devant implémenter l’interface ‘IPolicy’) aléatoire, greedy, ε-greedy, Boltzmann. Cependant seules les deux dernières ont réellement été utilisées par les agents pour l'apprentissage.

Nous avons dû rapidement anticiper le fait qu’un apprentissage d’un agent serait perdu à la fin d’une simulation, et nous a donc été nécessaire de gérer la sauvegarde des valeurs de la Qtable. La classe QExportImport permet de faire cela, par sérialisation ou désérialisation d'un objet java QLearning dans ou depuis un fichier.

Enfin, ayant décidé que les agents se partageraient leur table de Qvalues nous avons implémenté la classe QLearningFactory utilisant des méthodes statiques, permettant de gérer une instance de QLearning par classe à laquelle elle est associée. Ainsi en plus de permettre le partage des apprentissage entre agents, plus précisément entre ‘ExtAction’, cela permet aussi facilement d’étendre l’implémentation du Qlearning aux autres types d’agents du simulateur.

SIMON-FINE Thibaut
SÉNÉCHAL Sylvain

III. Travail réalisé

A. Présentation et modélisation du problème

Nous travaillons avec un environnement simplifié (carte par défaut modifiée) où les débris au sol ont été retiré, ainsi que les policiers, civils et ambulanciers. Nous travaillons donc uniquement avec les pompiers qui pourront circuler librement sur des routes sans débris. Le travail réalisé est cependant relativement facilement adaptable aux autres types d'agents pris isolément également, le problème ne devient complexe que lorsque tous les agents sont présents simultanéments.

Nous commençons par travailler avec un seul pompiers, puis avec 4 pompiers qui utiliseront et mettront à jour une même QTable globale.
Nous entraînons les agents sur au moins 5 simulations pour obtenir des résultats qui commencent à être significatifs.

Nous définissons l'état de l'agent avec 4 états :

Etat	Domaine de définition
Bâtiment en feu à portée	Binaire (0 / 1)
Quantité d'eau disponible	Ternaire (0 : vide / 1 : partiellement plein / 2 : plein)
Refuge proche	Binaire (0 / 1)
Être dans le refuge	Binaire (0 / 1)

Ces 4 informations d'état devraient être suffisantes pour que l'agent puisse prendre les meilleurs décisions.

Nous définissons 5 actions pour l'agent :

- Recherche de feu
- Se rendre à portée du plus proche bâtiment en feu
- Éteindre un feu
- Se rendre au refuge
- Remplir son réservoir d'eau

La fonction de reward est le point le plus important à définir dans notre modèle puisqu'une mauvaise reward induirait en erreur nos agents dans leurs prises de décisions.
Elle est définie de telle à éviter les non sens. La difficulté étant qu'il y a beaucoup de cas à envisager, ce qui implique beaucoup de code de type "If Then Else".

Par exemple :

- Éteindre un feu quand la réserve d'eau est à 0 => reward négative
- Remplir le réservoir quand il est plein => reward négative
- Partir à la recherche d'un feu quand la réserve d'eau est à 0 => reward négative

B. Résultats obtenus

Voici quelques valeurs remarquables de la QTable obtenue obtenue après entraînement :

Etat	Recherche	goBuilding	éteindreFeu	aller refuge	remplir eau
0010	+17	+16	+12	+24	-9
0011	+14	+2	+9	-4.5	+20
0211	+0	+2	-5	-20	-2
1200	+8	+8	+12	-1	-5

Interprétation :

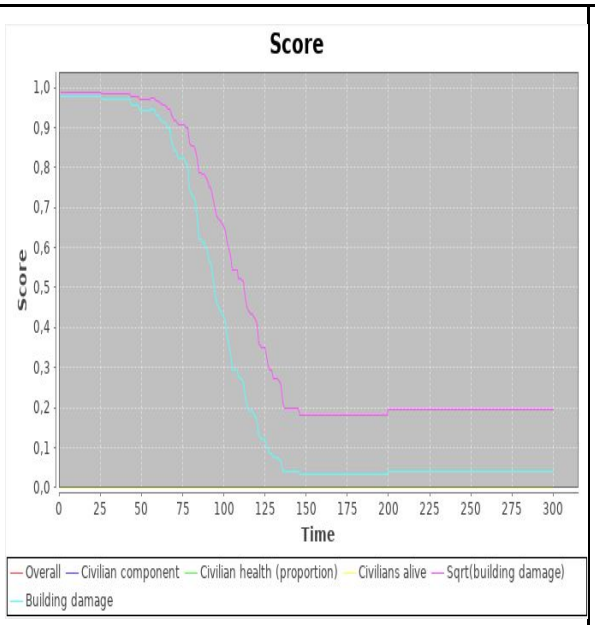
- Un agent qui manque d'eau et qui n'a pas de refuge à portée va se rendre au refuge
- Un agent qui manque d'eau et qui est dans un refuge va remplir son eau
- Un agent qui a plein d'eau mais pas de bâtiment en feu en vue va chercher un bâtiment en feu
- Un agent qui a plein d'eau et un bâtiment en feu à portée va tenter de l'éteindre

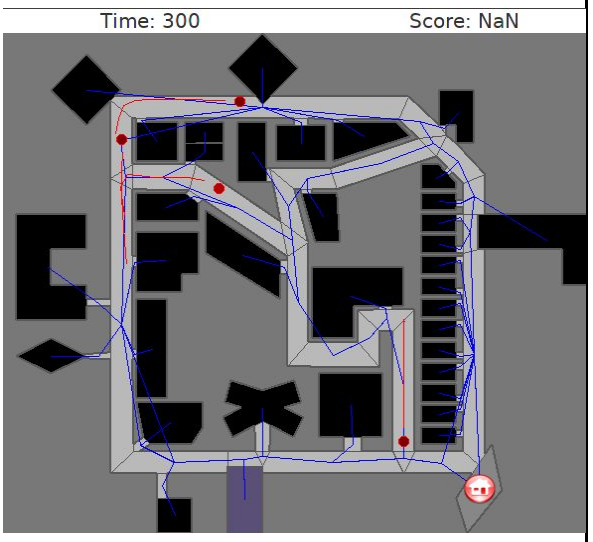
Score de destruction des bâtiments et état visuel de la carte

On commence par obtenir un score et un état de référence en faisant tourner la simulation avec des agents inactifs, et avec les agents basiques fournis dans sample ADF, voici ce qu'on obtient :

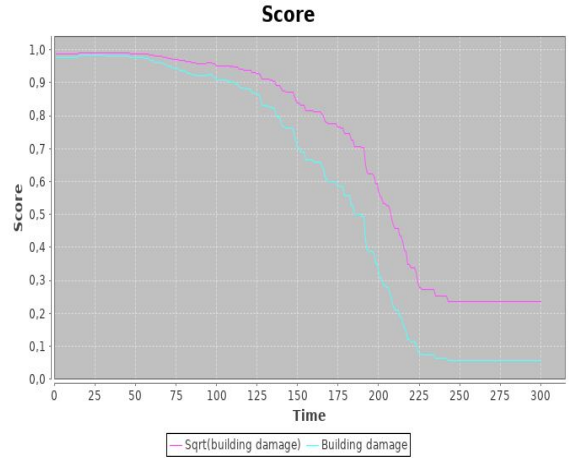
SIMON-FINE Thibaut
SÉNÉCHAL Sylvain

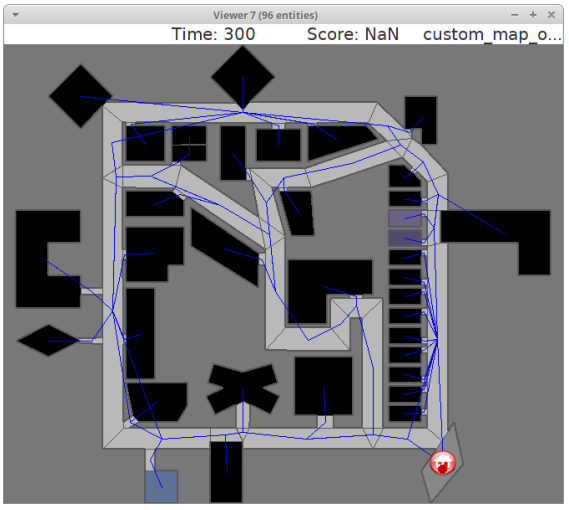
Référence 1 : Agents inactifs





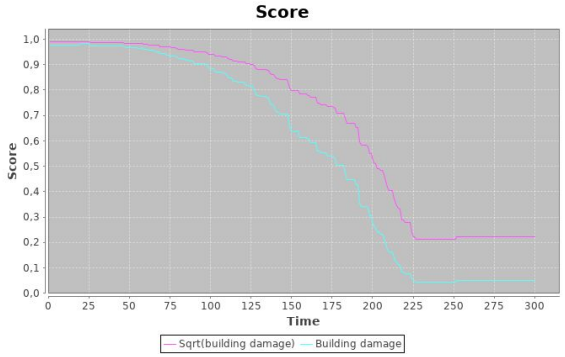
Référence 2 : Sample ADF, single agent

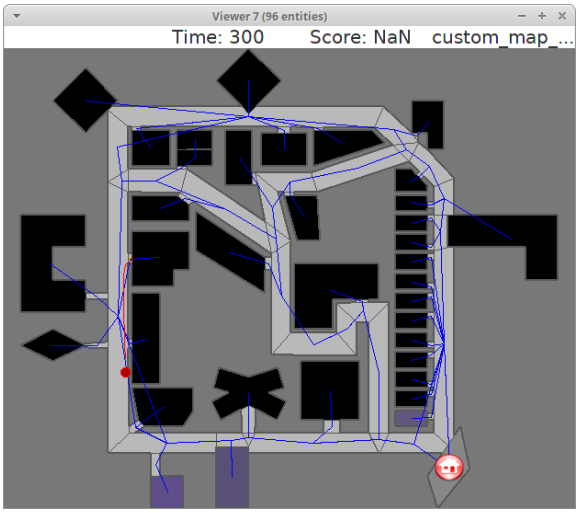




SIMON-FINE Thibaut
SÉNÉCHAL Sylvain

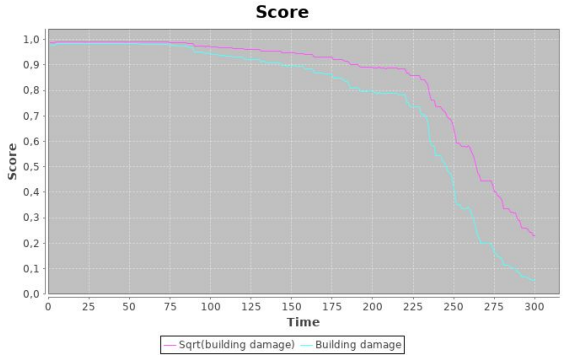
Agent **avant** apprentissage

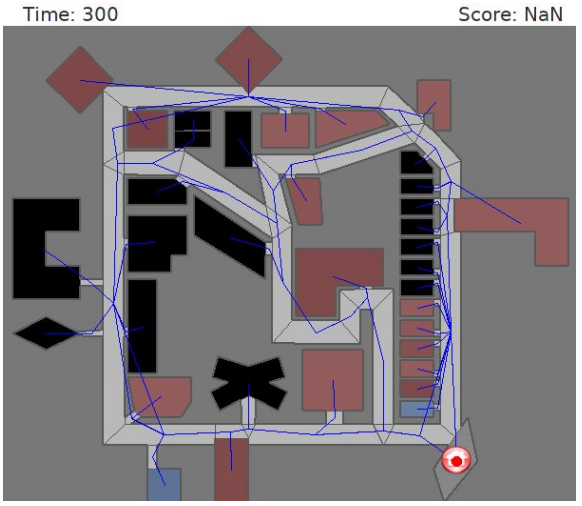




L’agent sans apprentissage obtient le même score que l’agent ADF référent

Agent **après** apprentissage

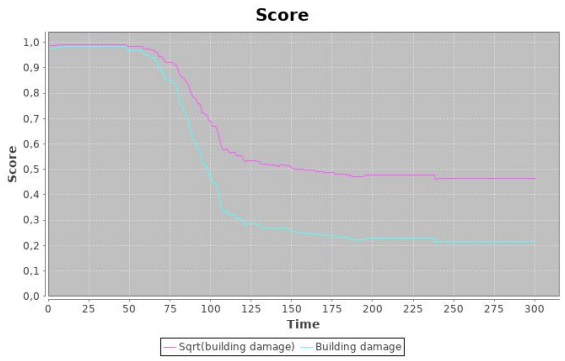
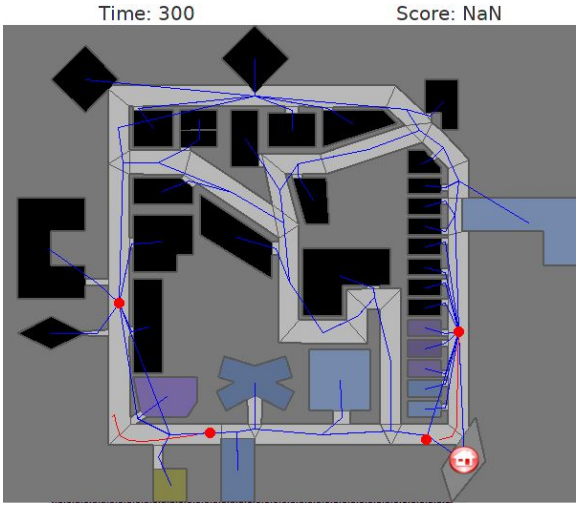


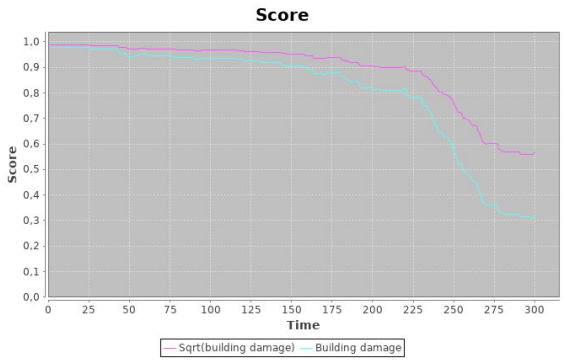
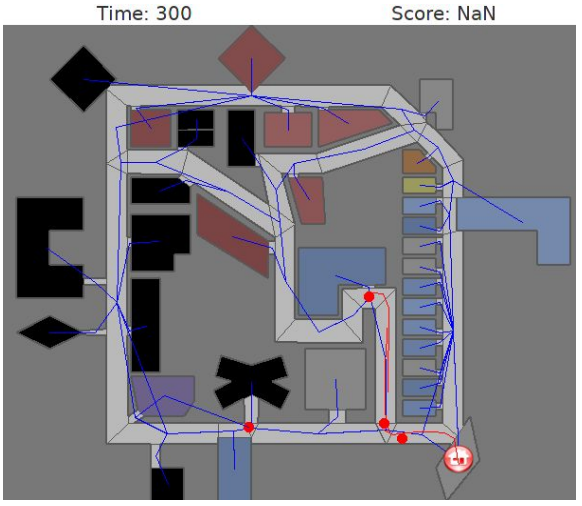


Observation : L’agent seul permet de ralentir significativement la destruction des bâtiments, mais il a trop de tâche à effectuer et se fait submerger par le nombre de feu à éteindre vers la fin

SIMON-FINE Thibaut
SÉNÉCHAL Sylvain

Score de destruction et état de la carte en Multi-Agents

4 Agents **au début** de l'apprentissage	
	
Observation : Les 4 agents sauvent quelques bâtiment très proches de leur réserve d'eau mais la plupart des autres bâtiments brûle dès le tick 100 de la simulation.	

4 Agents **après** apprentissage	
	
Observation : Ici les 4 agents ont réussi à sauver 30% des bâtiments sur la partie en bas à droite de la carte, proche de leur source d'eau. Les résultats sont bien meilleurs qu'avec un agent entraîné mais seul. Les bâtiments détruits ne le sont qu'après 200 ticks, car sur les premiers ticks ils ont parfois été éteints.	

SIMON-FINE Thibaut
SÉNÉCHAL Sylvain

C. Une extension

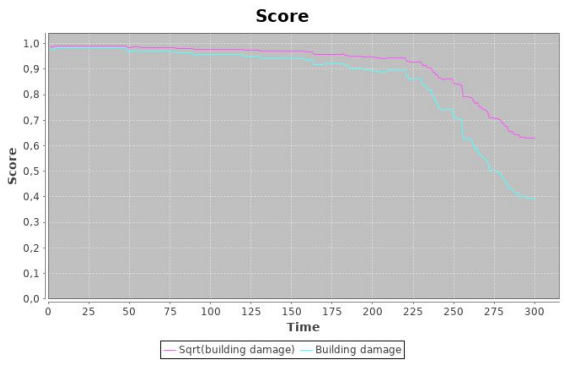
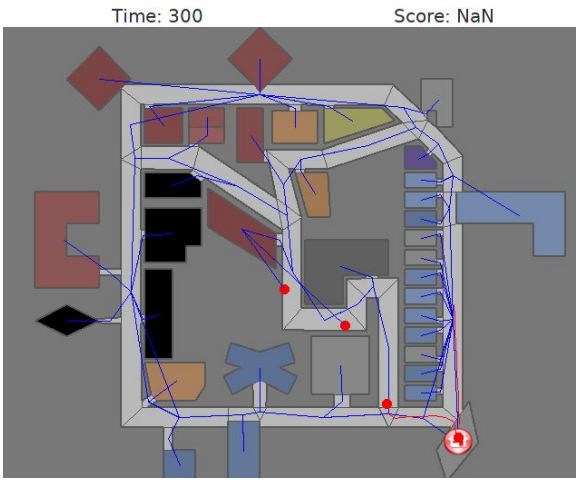
1. Nouveau paramètre

Bien que la carte soit de taille assez faible, nous avons remarqué que les agents restent souvent proches de leur point de départ et surtout de leur réserve d'eau, mais surtout, qu'ils ont tendance à tous travailler dans une même zone. Par conséquent il y a systématiquement toute une partie de la carte où les agents ne se rendent jamais et où les dégâts sont importants.

Pour régler ce problème, on a donc essayé un apprentissage en utilisant un paramètre supplémentaire pour la définition de l'état :

Etat	Domaine de définition
Bâtiment en feu connu	Binaire (0 / 1)
Bâtiment en feu à portée	*Binaire (0 / 1)*
Quantité d'eau disponible	*Ternaire (0 : vide / 1 : partiellement plein / 2 : plein)*
Refuge proche	*Binaire (0 / 1)*
Être dans le refuge	*Binaire (0 / 1)*

L'objectif recherché avec l'ajout de ce paramètre est de permettre aux agents ayant observé des feux (mais n'ayant pu les éteindre) lors d'exploration aient plus facilement tendance à retourner auprès de ces derniers pour les éteindre.

4 Agents **après** l'apprentissage	
	
Observation : Les agents ont sauvés plusieurs bâtiments et empêché la destruction complète d'autres, 90% lors de l'itération 200 et un peu plus de 40% en fin de simulation.	

SIMON-FINE Thibaut
SÉNÉCHAL Sylvain

Les agents obtiennent plus souvent de meilleurs résultats car ils ont une tendance plus forte à s'éloigner du point d'eau. Mais souvent cela ne suffit pas forcément à empêcher la création de foyers de feu qui peuvent provoquer les agents à devoir se retrancher près de leur source d'eau.

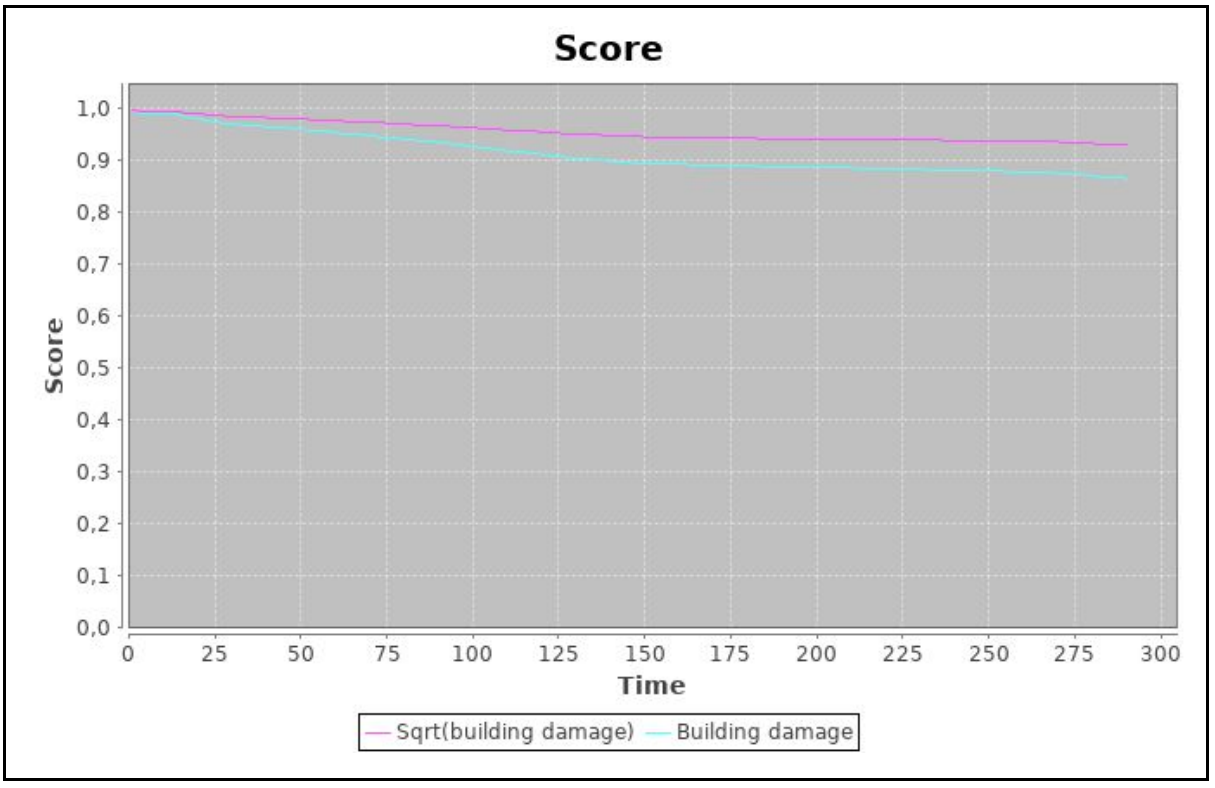
2. Autres amélioration

Pour améliorer la qualité de nos agents, on pourrait intégrer dans les états des agents, l'état des autres agents, pour avoir une idée de la position et des actions des autres agents, et mettre une reward négative quand des agents sont trop proches ou font la même tâche. Cela impliquerait donc aussi d'utiliser la l'aspect communication pour chaque agent puisse obtenir les informations sur les autres agents.

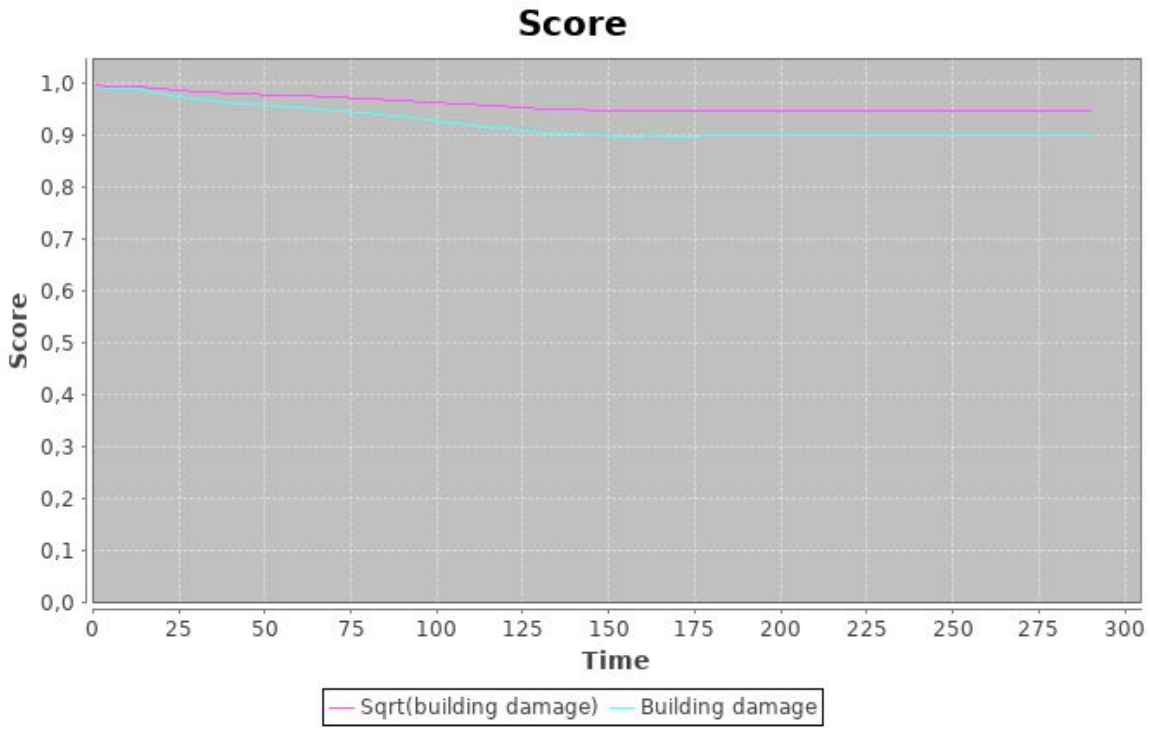
Evidemment on aurait un problème au niveau de la taille de la QTable qui serait démultipliée avec l'augmentation du nombre d'agents.

3. Test sur une grande carte

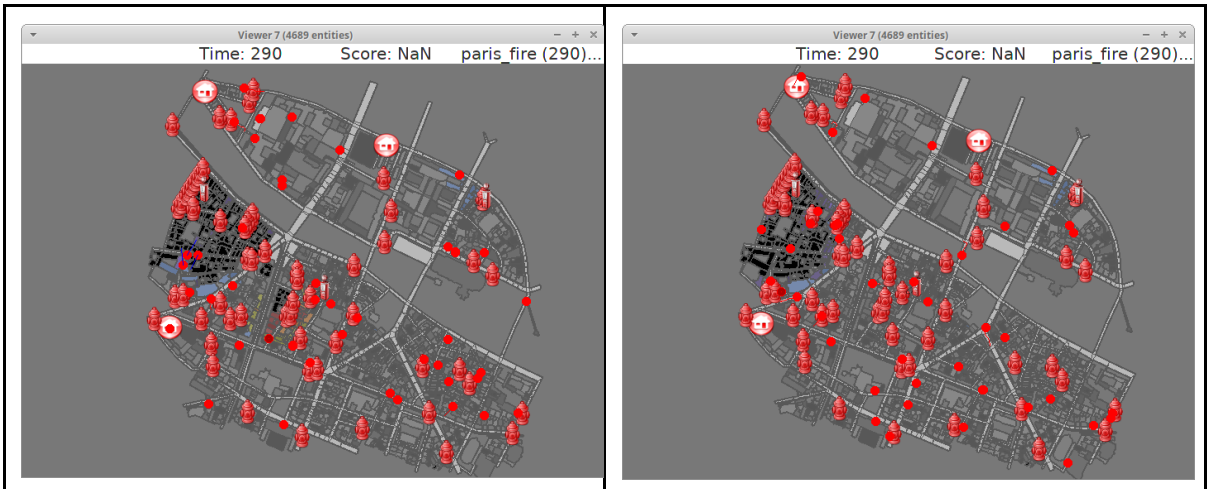
Nous avons voulu essayer la simulation sur une des grandes cartes proposées, bien que ces dernières nécessitent une plus grande puissance de calcul.
Nous avons choisi la carte Paris à laquelle nous avons retiré tous les éléments ne concernant pas les incendies. La carte possède donc quelques refuges et de nombreuses bornes incendie, ainsi que 46 agents pompiers. La simulation dure 290 ticks.



SIMON-FINE Thibaut
SÉNÉCHAL Sylvain



Comparaison des scores entre test_team (en haut) et les agents Qlearning (en bas)



Comparaison de l'état final entre test_team (gauche) et les agents Qlearning (droite)

On peut observer que l'équipe ayant utilisé l'apprentissage a été légèrement plus efficace, sa courbe de score descendant jusqu'à stagner vers 150; là où l'équipe de test voit sa courbe descendre continuellement (bien que légèrement). On peut observer cela sur la carte où le feu a été un peu mieux contenu dans un quartier par les agents ayant eu un apprentissage avec le Qlearning.

SIMON-FINE Thibaut
SÉNÉCHAL Sylvain

IV. Conclusion

À la suite des simulations effectuées, nous avons pu observer que les agents se concentrent vers les accès d'eau, car ils doivent y retourner pour remplir leur réservoirs; c'est la technique qui tend logiquement à se mettre en place pour maximiser la reward. Le feu se développe alors plus fortement dans les zones éloignées et cela crée rapidement des foyers importants qui s'étendent ensuite aux bâtiments proches. Ces observations du regroupement auprès des points d'eau, permet sans doute de montrer aussi l'intérêt des bornes incendies à intervalles régulier afin de rapprocher les sources d'eau des feux potentiels. Par ailleurs, introduire une reward permettant aux agents de se répartir sur la carte uniformément pourrait résoudre en partie ce problème.