Concours drone - Rapport

Essaims de drones autonomes facilitant la traversée d'un passage piéton dans un contexte de ville intelligente

Étudiants : Lucas Monlezun et Axel Mourlanne

Année universitaire : 2021 - 2022

Table des matières

1 Introduction au projet

Le projet a été pensé pour l'UE « Concours drone », dans laquelle il est demandé de proposer une solution innovante basée sur un drone ou un essaim de drones, afin de répondre à une problématique choisie.

Notre projet s'inscrit dans le cadre d'une zone urbaine, et plus particulièrement dans une ville intelligente. Dans les zones urbaines, il est difficile d'assurer sans problème le partage de la route entre les usagers.

Il y a régulièrement des accidents entre des voitures et des piétons, qui sont souvent liés à de mauvais comportements de la part des automobilistes. En particulier, les automobilistes arrivent parfois trop vite aux abords d'un passage piéton et ne voient que trop tard les piétons traverser, ce qui peut créer un accident. Dans certaines situations, il peut aussi arriver qu'un piéton soit masqué par divers obstacles au moment où il commence à traverser (comme par exemple des voitures garées sur le côté), et les automobilistes ont alors du mal à le voir, ce qui crée une situation dangereuse.

Dans la majorité des cas, même s'il n'y a pas d'accident, les piétons ont peur de s'engager sur la route alors même qu'ils ont la priorité en traversant par un passage piéton, et sont obligés d'attendre que les voitures s'arrêtent sur les deux voies.

Il nous paraissait donc nécessaire de trouver des solutions afin de sécuriser au maximum les passages piétons.

En effet, il est possible d'utiliser des drones pour permettre à des piétons de traverser un passage piéton de manière plus sécurisée et plus sereine. Pour cela, à l'aide de bornes situées de chaque côté du passage, le piéton appuierait sur un bouton situé sur la borne pour « appeler » des gardes de traversée, autrement dit des drones disséminés dans la zone urbaine utilisant ce système de traversée intelligente.

Ces drones arriveraient donc au niveau de la route afin d'une part d'alerter les voitures de la traversée qui s'apprête à se faire grâce à des LEDs clignotantes bleues et rouges, et d'autre part de dissuader les voitures à passer malgré tout, comme bien des conducteurs font lorsque le piéton est toujours sur la voie opposée.

Cette dissuasion passe par le blocage des deux voies de chaque côté du passage piéton par des gardes de traversée imposants et donc capables d'abîmer la voiture d'un usager refusant de s'arrêter.

Ainsi, un piéton peut s'engager sur la route de manière plus sereine car il sait que les usagers de la route s'approchant sont conscients de sa traversée. Une fois cette dernière achevée, les gardes repartent.

Dans le cadre d'une ville intelligente, il faudrait faire en sorte que les drones soient répartis dans toute la ville, au niveau de stations sur lesquelles ils se rechargent. Les boutons pressés par les piétons seraient connectés, ce qui fait que lorsqu'un piéton appuie dessus, la ville intelligente (en l'occurrence la station de contrôle) pourrait affecter les drones disponibles (et disposant de suffisamment d'autonomie pour assurer une mission classique) les plus proches au passage piéton correspondant, et ce de manière complètement autonome et dynamique.

Il serait aussi possible d'affecter les drones à des missions récurrentes, comme par exemple les entrées et les sorties d'école le matin et le soir.

2 Description et analyse de l'existant

2.1 Drones compagnons

Notre projet comporte des similitudes avec le principe de drone « compagnon ». Le principe de ces drones est d'accompagner une personne afin notamment d'assurer sa sécurité, et ce de manière autonome. Ils peuvent être utiles par exemple la nuit, pour éviter les agressions ; le drone peut éclairer toute la surface située autour d'une personne, et peut également directement signaler une agression à la police pour que celle-ci intervienne plus rapidement.

Un drone compagnon peut consister en un drone personnel, qui accompagne tout le temps la même personne. Mais dans le cadre d'une ville intelligente, il peut aussi y avoir un réseau de drones compagnons ; lorsqu'une personne en a besoin, elle peut appeler un drone via une application, et le drone se rend alors à la position GPS de l'appelant. Les drones peuvent aussi se relayer au cas où la mission dure plus longtemps que leur autonomie.

2.2 Drones en milieu urbain

Outre les drones compagnons, plusieurs applications futures existent pour des drones en milieu urbain.

L'une de ces applications est la maintenance des infrastructures d'une ville. Un essaim de drones est capable de parcourir une ville afin d'inspecter les routes, les bâtiments, ou tout type d'infrastructure, et de détecter des défauts qui nécessitent une intervention de la part des services communaux. Un projet de cette nature est actuellement développé par l'Université de Leeds [1].

Une autre application serait le transport de personnes, ce qui permettrait de soulager le trafic routier et les transports publics à saturation dans les zones urbaines denses. Les drones permettraient de se déplacer beaucoup plus rapidement à travers une ville, et ce de manière plus écologique que les transports classiques. Plusieurs « taxis volants » de ce type existent déjà et sont en phase de test, tels que le CityAirbus [2], les drones Volocopter [3], ou encore le Lilium Jet [4].

Cependant la présence de drones en milieu urbain pose des problèmes, notamment de par le danger que cela peut engendrer. En effet, les drones ne doivent pas voler trop bas, au risque de rentrer en collision avec un obstacle et de tomber, ce qui serait très dangereux. Il faut aussi faire en sorte que les drones ne se percutent pas entre eux, ce qui nécessite des systèmes de communication fiables. Pour éviter au maximum ces problèmes, il est probablement nécessaire de définir des couloirs aériens exclusivement réservés aux drones, et avec des règles très strictes afin que celles-ci puissent couvrir toutes les situations possibles, mais ces règles ne sont pas encore clairement définies. Des travaux de recherche sont actuellement en cours afin de définir clairement ces couloirs et les règles qui s'y appliquent, par exemple dans [3].

1. Site officiel du projet « *Self Repairing Cities* »
2. Page du CityAirbus sur le site d'Airbus
3. Site officiel de Volocopter
4. Site officiel du Lilium Jet

2.3 Communications dans un essaim de drones

Dans le cadre d'une mission réalisée par un essaim de drones, il est nécessaire qu'il y ait un haut niveau de coopération entre les drones et/ou avec une station sol, afin que la mission se déroule convenablement. Par exemple, cette coopération peut simplement permettre aux drones de ne pas se heurter en plein vol.

Il existe au moins quatre types d'architectures de communication pour un essaim de drones [2] :

- Communication centralisée. Les drones sont directement connectés à une station sol, et ne sont pas connectés entre eux ; les messages transitent uniquement via la station sol.
- Communication par satellite. Les drones ne sont ni connectés à la station sol, ni connectés entre eux, et tous les messages transitent par un satellite qui les transmet au destinataire.
- Communication semi-centralisée. Plusieurs cellules sont déployées sur l'ensemble du territoire de la mission, chaque cellule contenant des drones et une station sol. Les drones d'une même cellule peuvent directement communiquer entre eux, mais ils doivent passer par la station sol pour communiquer avec les autres cellules.
- Communication ad hoc. Chaque drone est un noeud du réseau, et lorsqu'un message doit être transmis d'un drone à un autre ou à la station sol, alors celui-ci est transmis de proche en proche jusqu'à ce qu'il arrive au destinataire.

Les deux premières architectures ont des limitations qui les rendent difficilement utilisables dans le cadre de notre projet. La limitation majeure est la latence élevée entre l'envoi et la réception d'un message, due au fait qu'ils doivent forcément transiter par un système centralisé ; or, pour notre projet, ces messages auront notamment pour but d'éviter les collisions entre les drones, ils doivent donc être communiqués le plus rapidement possible afin d'anticiper ces potentielles collisions. De plus, des obstacles situés autour de l'émetteur pourraient empêcher la bonne transmission du message.

Les deux autres architectures sont beaucoup plus adaptées à notre projet par le simple fait qu'elles permettent à des drones proches les uns des autres de communiquer directement entre eux, ce qui réduit fortement la latence entre l'envoi et la réception.

Une autre spécificité de la communication dans le cadre d'un essaim de drones, est que cette communication doit se faire de manière résiliente, c'est-à-dire que la perte d'un drone ou des communications entre deux drones ne doit pas entraîner un échec de la mission [1]. Cette résilience se base sur l'utilisation de primitives de communications résilientes, c'est-à-dire des primitives qui ne doivent rien supposer sur l'environnement des drones. Dans ce cas, la meilleure solution est d'envoyer les messages en *broadcast*, c'est-à-dire que les messages sont envoyés tout autour du drone émetteur, en espérant que les drones suffisamment proches le reçoivent.

Cela dit, l'émetteur ne saura jamais si son message a bien été reçu ; il doit donc pouvoir continuer sa mission malgré cette incertitude.

3 *User stories et besoins*

3.1 *User stories*

Les *user stories* sont une suite de récits courts décrivant le fonctionnement d'un système ou d'une fonctionnalité. Leur objectif est de placer l'utilisateur – et en particulier ce qu'il attend du produit – au coeur du développement car, finalement, quelle que soit la complexité de ce produit, son succès dépendra principalement de la satisfaction des utilisateurs

Les *user stories* décrivent donc le bon fonctionnement du système du point de vue de l'utilisateur, avec des termes simples. De cette manière, les développeurs sont capables de tenir un cap et de se rappeler à tout moment des objectifs finaux d'un projet.

Au cours de ce projet, nous avons ainsi été amenés à réfléchir comme un utilisateur du système de traversée intelligente et cet exercice nous a permis d'imaginer des tâches auxquelles nous n'aurions peut-être pas pensé en réfléchissant d'abord à des besoins.

Il y a d'ailleurs un parallèle intéressant entre les *user stories* et les besoins car il est bien plus aisé d'identifier toutes les besoins et exigences non fonctionnels du projet lorsque nous nous mettons à la place d'un utilisateur.

Finalement, nous nous sommes accordés à choisir ces quelques *user stories* pour notre projet de Concours drone :

- **Le piéton bénéficie rapidement d'une équipe de drones pour traverser.**
  Cette exigence très simple est en réalité primordiale pour la simple et bonne raison qu'aucun piéton n'utiliserait le système s'il fallait une quarantaine de secondes pour que les drones arrivent.
- **Le piéton n'est pas gêné par les drones placés de part et d'autre du passage piéton.**
  Il est important que les drones n'empiètent pas sur le passage car un utilisateur pourrait avoir peur d'être blessé.
- **La traversée du piéton est connue des usagers à proximité grâce aux drones qui les alertent via des LEDs.**
  Un aspect très important de ce système est la dissuasion. Pour que cette dernière soit effective, les gardes de traversée se doivent d'être visibles.
- **Le piéton doit pouvoir bénéficier de plusieurs drones à tout instant de la traversée, quelle que soit leur autonomie.**
  Il est inconcevable qu'un utilisateur puisse être abandonné par les drones avant la fin de la traversée. S'il doit y avoir des remplacements à cause d'une autonomie non suffisante, cela doit se faire progressivement.

La définition de ces *user stories* nous a permis de déduire plusieurs besoins fonctionnels et non fonctionnels, que nous détaillons ci-dessous.

3.2 Besoins fonctionnels

- **Les drones doivent se trouver à leur emplacement attribué.**

Pour cela la station de contrôle doit s'assurer que le nombre exact de drones nécessaires soit mobilisé et que ces drones n'aient pas déjà un emplacement attribué. Ainsi, ils sont placés de part et d'autre du passage et le piéton peut aisément et sereinement circuler.

— **Les drones doivent pouvoir alerter les usagers de la route des différentes étapes de la mission en cours.**
Afin d'être visibles par les usagers mais également de dissuader ces derniers d'ignorer le piéton, les drones doivent être répartis le long du passage piéton. De cette manière, ils peuvent être vus de loin par les voitures et ces dernières n'ont pas la place de contourner les drones. L'essaim de drones doit également utiliser une alternance de couleurs vives lors de la phase principale de détection (bleu et rouge), et la couleur verte lorsque la mission s'apprête à prendre fin.

— **Le piéton doit être accompagné de plusieurs drones à tout instant de la traversée, quelle que soit leur autonomie.**
Le piéton ne doit jamais être délaissé par les drones une fois qu'il a commencé à traverser la route. Si un drone n'a plus suffisamment de batterie pour continuer la mission, la station de contrôle doit faire en sorte de le remplacer. De plus elle ne doit répondre qu'à une requête de remplacement à la fois afin d'avoir en permanence plusieurs drones sur une mission.

— **Les drones doivent éviter toute collision avec un obstacle ou entre eux.**
Ils doivent être équipés de systèmes permettant de détecter la présence d'un obstacle (tels que des immeubles, des voitures, ou d'autres drones), et adapter leurs mouvements afin d'éviter tous ces obstacles.

— **Les drones doivent pouvoir communiquer entre eux.**
Un protocole de communication entre les drones doit être défini, afin que ceux-ci puissent s'envoyer différents types de messages. Ces messages, notamment des messages indiquant la position des drones, peuvent permettre en particulier d'éviter les collisions entre eux.

3.3 Besoins non fonctionnels

— **Les drones doivent arriver au passage intelligent le plus rapidement possible.**
Pour ce faire, seuls les drones les plus proches doivent être sélectionnés, à la condition qu'ils soient libres et qu'ils aient suffisamment de batterie.

— **Un essaim de drones ne doit pas gêner les usagers de la route plus qu'il n'est nécessaire.**
En l'occurrence, les drones doivent n'accorder qu'un certain délai en début de mission sans aucune détection de piéton afin de prévenir des faux positifs. Il faut également limiter au maximum le temps des drones au passage lorsque la mission est finie afin de permettre à nouveau une circulation normale des voitures.

— **Les drones doivent avoir une autonomie acceptable.**
Même avec un système dynamique de remplacement des drones lorsque l'un d'eux n'a plus suffisamment de batterie, il est nécessaire que chaque drone puisse rester

au minimum une minute sur un passage pour une mission classique afin de ne pas avoir constamment un drone manquant à la mission en raison de remplacements incessants.

4 Choix d'implémentation

4.1 Simulation

Le projet n'est pas réalisable avec du vrai matériel, puisqu'il nécessiterait d'avoir à disposition une ville intelligente (ou un système équivalent) sur laquelle réaliser nos tests. A la place, nous avons créé une simulation qui doit prendre en compte le maximum de problématiques qui pourraient être rencontrées dans la vraie vie, comme par exemple la répartition des drones dans la ville, le comportement des drones face à des véhicules qui ne s'arrêtent pas, ou encore la gestion des faux positifs (i.e. quelqu'un appuie sur un bouton mais personne ne traverse).

Dans le cadre de la robotique et des drones, il existe plusieurs solutions logicielles pour réaliser des simulations.

Une solution répandue est l'utilisation conjointe de Robot Operating System (ROS) et Gazebo. ROS est un « ensemble de bibliothèques logicielles et d'outils aidant à la création de logiciels pour la robotique » [5] ; ROS peut donc permettre de créer l'application qui sera exécutée en temps réel sur nos drones. Gazebo est quant à lui un logiciel de simulation de robots articulés, et peut notamment simuler de manière réaliste la physique des robots. Il est possible d'intégrer ROS à Gazebo afin de contrôler les drones présents dans la simulation grâce aux données envoyées par l'application ROS.

Une autre solution est d'utiliser Unity. Il s'agit d'un moteur de jeu et d'un logiciel permettant de faire des jeux en 2D ou en 3D ; il est très répandu dans le domaine du jeu vidéo, mais est aussi régulièrement utilisé pour réaliser des simulations impliquant des robots ou des drones.

Pour notre projet, nous utilisons Unity car cela semble la solution la plus adaptée. En effet, notre projet implique de devoir modéliser et simuler une ville intelligente, ce qui est très facile avec Unity mais beaucoup moins avec Gazebo qui n'est pas fait pour ce genre de simulation. De plus, Gazebo est particulièrement utile dans le cas où nous souhaitons simuler de façon réaliste la physique des drones ; or, il ne s'agit pas de la problématique la plus importante de notre projet. Ici, il est plus important de gérer la communication entre les drones, ainsi que la façon dont la ville intelligente gère les drones ; Unity permet d'implémenter cela de manière simple et réaliste.

En revanche, ROS aurait été probablement très utile pour gérer le comportement individuel de chaque drone (notamment la façon dont ils détectent un piéton ou une voiture) ; mais il est aussi possible de gérer cela avec Unity, et nous avons considéré que la simulation de la ville intelligente était un élément très important du projet et qu'il aurait été trop compliqué de l'implémenter avec Gazebo.

Enfin, nous avions déjà des connaissances sur Unity avant le projet, contrairement à ROS et Gazebo. Or, au vu des délais serrés pour le réaliser, il aurait été sûrement trop

5. Site officiel de ROS

long d'apprendre à utiliser de nouveaux logiciels et nous n'aurions pu implémenter qu'une petite partie des aspects du système de traversée que l'on souhaite simuler.

4.2 Architecture

L'architecture de notre projet correspond à l'architecture classique d'un projet Unity. En particulier, le dépot est composé de plusieurs dossiers différents :

– **'Assets/'** est le dossier comprenant la quasi-totalité du projet. C'est dans ce dossier que se trouvent tous les scripts, les *prefabs* (les modèles de la ville, du personnage incarné lors de la simulation, des voitures ou encore des drones), les *materials* ou encore le contenu de la scène correspondant au scénario.

– **'Build/'** contient les exécutables et les fichiers nécessaires à leur lancement.

– **'Logs/'** est le dossier comprenant toutes les traces de messages, d'avertissements ou d'erreurs apparus dans l'application.

– **'UserSettings/'** contient les préférences de l'utilisateur dans l'éditeur.

– **'Packages/'** contient une multitude d'éléments compressés du projet nécessaires au lancement de la simulation dans l'éditeur.

– **'ProjectSettings/'** contient quant à lui de nombreux fichiers de configuration pour le projet projet.

– A la base de cette arborescence de fichier se trouve '**README.md**' dans lequel se trouvent toutes les indications permettant notamment de lancer la simulation.

Le dossier le plus intéressant est **'Assets/'**, à l'intérieur duquel se trouvent les sous-dossiers suivants :

– **'Drone/'** contient tous les éléments composant un drone.

– **'Simple city plain/'** contient tous les éléments de base permettant de créer une ville (immeubles, rues, passages piétons, etc.).

– **'Prefabs/'** contient certains éléments spécifiques à notre simulation (bornes, boutons, etc.)

– **'Materials/'** contient les *materials* Unity appliqués à certains objets de la scène.

– **'Scenes/'** contient les fichiers qui définissent notre scène.

– **'Scripts/'** contient l'ensemble des scripts appliqués aux éléments de notre simulation. Ce dossier comprend également la classe ***Message***, qui définit la structure et le type des messages envoyés par les drones.

– **'Tests/'** contient les fichiers de tests unitaires.

4.3 Présentation des éléments de la simulation

4.3.1 Système de traversée intelligent

Pour commencer, la quasi-totalité du projet se base sur le système de traversée de passage piéton intelligent, que voici :

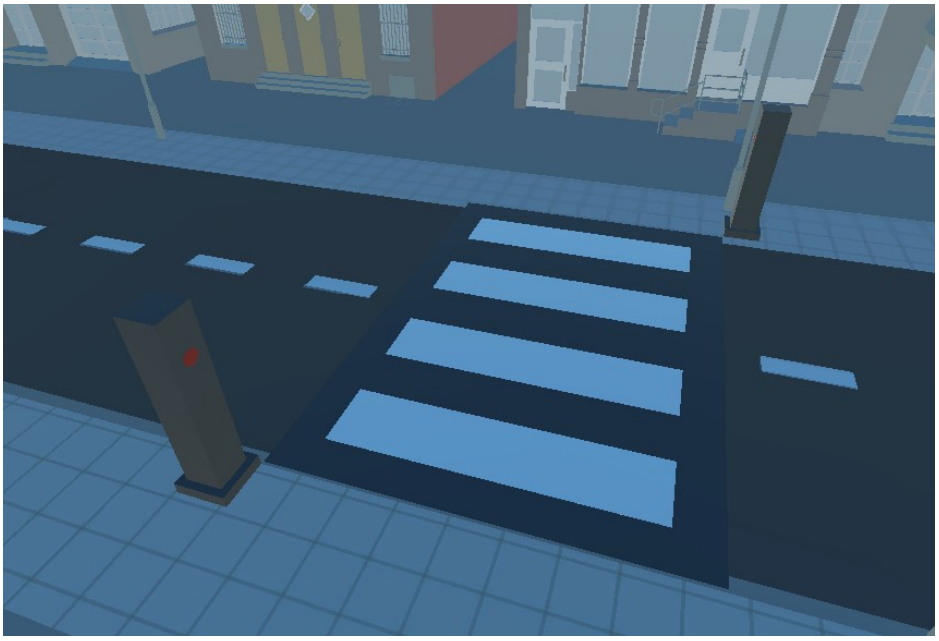

FIGURE 1 – Système de traversée intelligent

Il se compose d'un passage piéton classique accompagné d'une borne de chaque côté de la route et des six emplacements invisibles placés de manière relative au passage. En pressant le bouton rouge de l'une des bornes, la station de contrôle des drones est alertée et envoie les gardes.

Ce système intelligent d'aide à la traversée est un prefab appelé ***crossingSystem*** dans Unity.

4.3.2 Gardes de traversée

Les gardes de traversée sont des drones désignés par la station de contrôle pour qu'ils se rendent sur le lieu de la mission, en l'occurrence au passage piéton associé à la borne utilisée par un utilisateur.

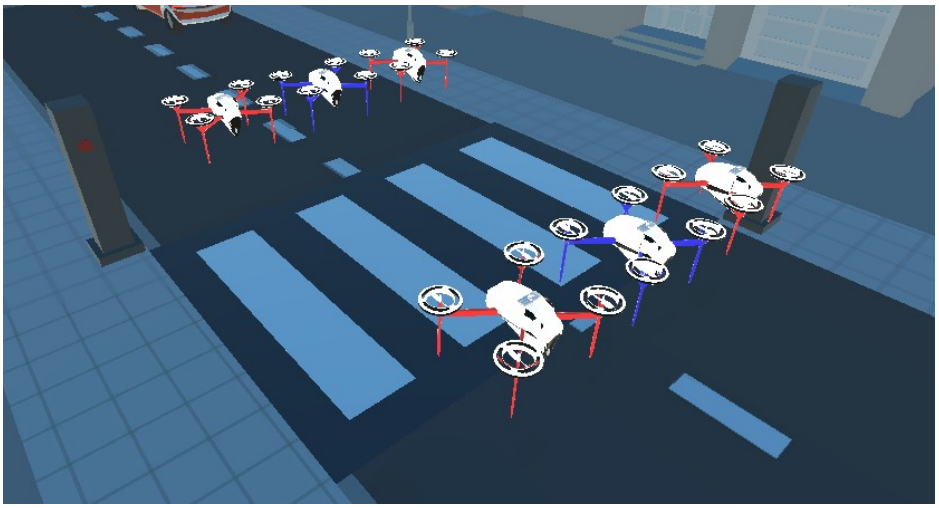

FIGURE 2 – Essaim de drones en charge d'assurer la sécurité des piétons

Ces drones sont destinés à survoler un passage piéton afin d'assister et protéger les usagers en constituant un essaim. Ils sont autonomes et prennent des décisions en tant qu'équipe.

Ils arborent différentes couleurs au cours de la mission.

Lors de l'étape cruciale pendant laquelle un ou plusieurs piétons traversent la route, les drones clignotent en émettant alternativement une lumière bleue et rouge.

Pour détecter des piétons, les gardes envoient une multitude de rayons dans des directions différentes afin de couvrir l'intégralité du passage :

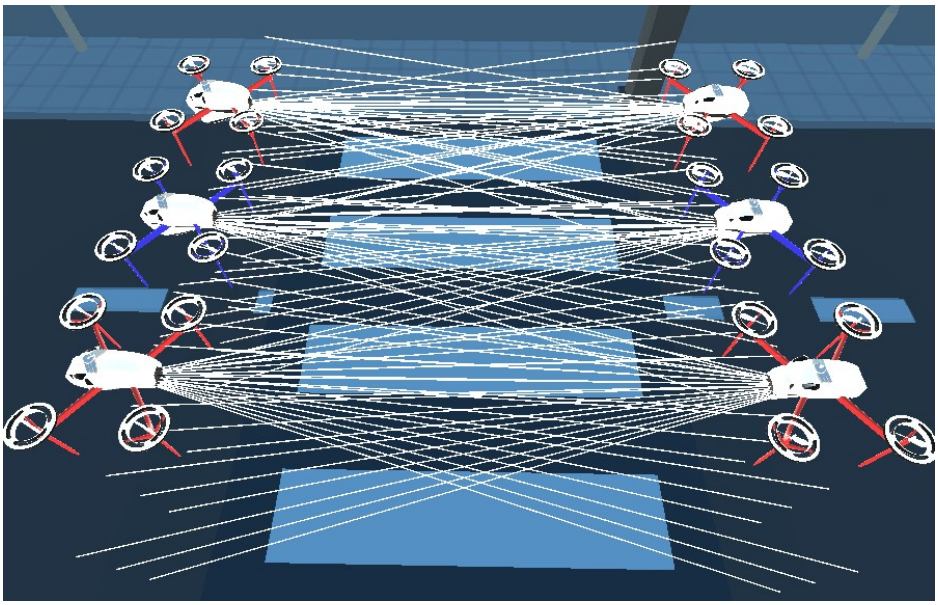

FIGURE 3 – Rayons de détection utilisés par les gardes

Ici les rayons ont exceptionnellement été rendus visibles. L'espace entre les rayons est bien trop petit pour qu'une personne n'y échappe et même s'il s'agit d'enfants : plusieurs rayons sont également émis avec un angle incident au sol.

Il est aisé de se rendre compte qu'un piéton n'est plus détecté par les drones au moment exact où il quitte le passage piéton.

Lorsque la mission se termine, les drones émettent cette fois-ci une lumière verte afin d'avertir les voitures de cette fin de mission.

FIGURE 4 – Gardes en train de quitter le passage piéton

Enfin, si un drone doit être remplacé durant la mission (par exemple si son autonomie est trop faible), il arbore la couleur noir pendant quelques secondes avant de laisser sa place à son successeur.

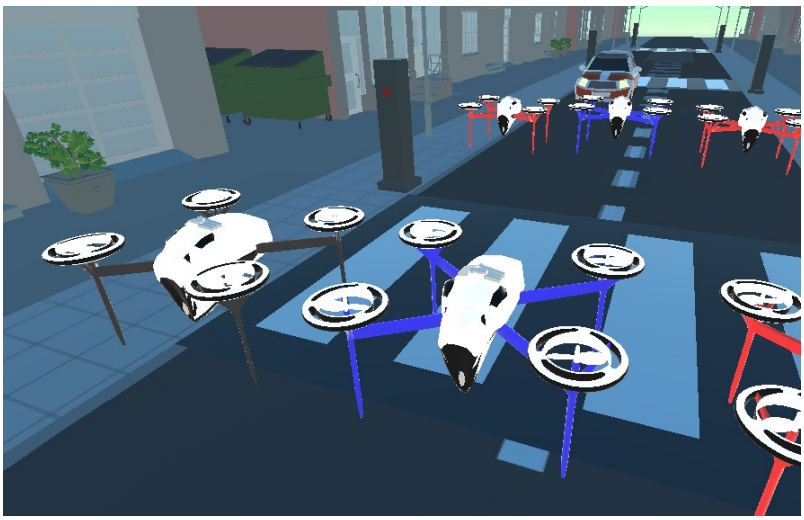

FIGURE 5 – Garde sur le point d'être remplacé (à gauche)

4.3.3 Voitures

Pour accompagner la simulation du système de traversée, il nous semblait pertinent de faire appel à des voitures, puisqu'elles font partie des personnages principaux dans ce scénario.

FIGURE 6 – Voiture circulant dans la ville

C'est en effet pour protéger les piétons des véhicules – en particulier des voitures – qu'existe ce système de traversée intelligente.

Ainsi, dans cette situation, chaque voiture peut grâce au script ***Car.cs*** détecter tous types d'objets à une distance de dix mètres. Dès lors qu'un objet, qu'il soit un drone ou un piéton, est détecté, le véhicule va décélérer jusqu'à un arrêt complet.

Cette détection se fait sur toute la voie sur laquelle la voiture se trouve et également en hauteur pour détecter un drone en train d'arriver pour une mission.

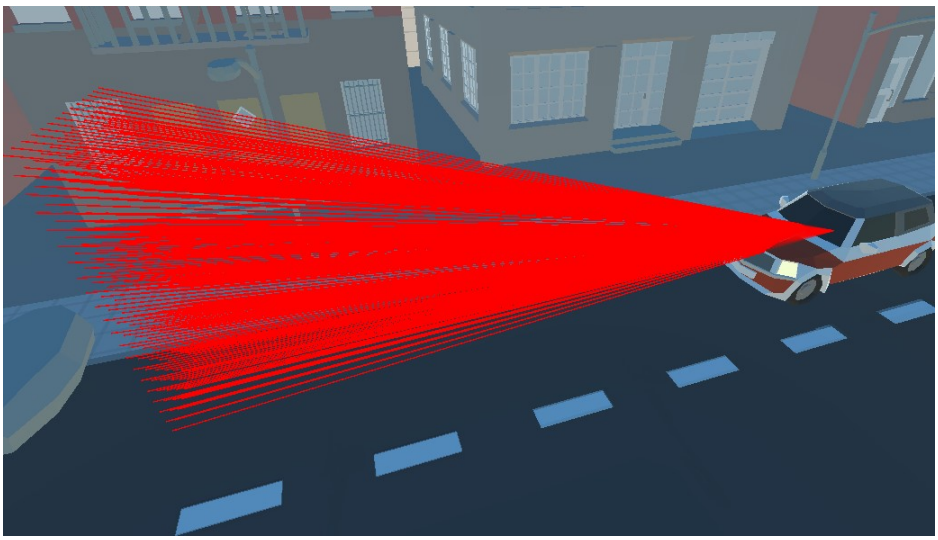

FIGURE 7 – Champ de vision d'une voiture dans la simulation

Dans cette simulation, une voiture est également capable de reculer si jamais elle bloque le positionnement d'un des gardes participant à une mission.

Nous avons voulu rendre la simulation la plus réaliste possible afin d'envisager un maximum de possibilités et de problématiques qu'un tel système pourrait rencontrer

dans la vie réelle, dont le fait qu'un conducteur s'apercevant que son véhicule gêne la mission devrait reculer s'il en a l'occasion.

4.3.4 Piéton

Dans cette simulation, de la même manière que les voitures, le piéton est le personnage principal du scénario. Le système de traversée intelligente a été créé pour lui, et c'est également lui qui décide d'appeler un essaim de gardes ou non.

FIGURE 8 – Piéton pouvant utiliser le système de traversée intelligent

Afin d'incarner ce piéton, nous lui avons attaché le script ***Human.cs*** et lui avons fixé la caméra au niveau de sa tête. Ainsi, nous obtenons une vue à la première personne.

Pour le faire avancer nous avons choisi la touche 'W', et les flèches directrices ← et → pour modifier l'orientation du piéton et donc la vue de la caméra.

5 Implémentation des fonctionnalités

5.1 Arrivée des drones

5.1.1 Boutons et tags

Comme précisé précédemment, dans le cadre de la simulation sur Unity, le système de traversée intelligente se compose d'un passage piéton, de deux bornes sur le trottoir de chaque côté et de six positions pour les drones relatives aux bandes composant le passage.

Sur chaque bouton des bornes est attaché le script ***Button.cs*** trouvable au chemin ***Assets/Scripts/Button.cs***. De cette manière, en pressant le bouton de l'une des bornes, la méthode ***OnMouseDown*** est appelée. Le tag du bouton étant une chaîne de caractères décrivant un nombre, la méthode ***StartMission*** de la station de contrôle (***ControlStation.cs***) est alors appelée avec ce nombre en paramètre. Dans un contexte

de ville connectée, on peut supposer que c'est grâce au réseau 5G que la borne peut « demander » à la station de démarrer une mission.

Cette station de contrôle possède une liste de toutes les positions de chaque emplacement, et ce pour chaque passage piéton. De plus ces collections de positions sont classées en fonction du tag desdites positions dont les derniers caractères sont les mêmes que ceux des boutons associés.

Ainsi, en passant le tag du bouton en paramètre de ***StartMission***, la station de contrôle est capable d'assigner directement à un drone une position pour la mission.

5.1.2 Choix des drones pour la mission

Lorsqu'une mission se lance, il est primordial de s'assurer que suffisamment de gardes puissent être affectés à celle-ci : c'est là l'utilité de la méthode ***AutonomyCheck*** qui va renvoyer le booléen *true* s'il y a assez de drones inactifs tels que leur autonomie permette d'effectuer une mission classique.

Une autre vérification faite avant de lancer la mission est de s'assurer qu'une équipe de gardes n'est pas déjà assignée à ce passage.

Cela peut être vérifié grâce à l'attribut ***currentlyUsedSpots*** de la station de contrôle qui est simplement une liste de l'ensemble des positions actuellement occupées par un drone, et ce pour chaque passage. C'est donc une liste de listes de positions.

Cela permet entre autres d'empêcher un piéton d'appeler des drones alors qu'un autre piéton l'a déjà fait sur le trottoir opposé.

Une fois que ces deux vérifications sont faites, la liste des drones (ou du moins une copie) est triée en fonction de leur distance par rapport au piéton qui vient d'appuyer sur le bouton.

De cette manière, la distance parcourue par les drones au début et à la fin de la mission est minimisée, ce qui leur permet de minimiser également le temps nécessaire au trajet et donc de conserver la charge de leur batterie.

Ainsi, les drones les plus proches et ayant suffisamment d'autonomie pour compléter une mission classique sont mobilisés.

5.1.3 Sélection d'un chef d'équipe

Le premier drone de la liste est en théorie choisi pour être le chef d'équipe des gardes mobilisés pour la mission.

Cela signifie notamment qu'il va être responsable de la coordination de l'équipe durant toutes les étapes de la mission. Par exemple, les drones ne vont débuter la détection de piétons et l'allumage des LEDs qu'une fois que tous les gardes de la mission sont placés.

C'est également ce chef d'équipe qui va comptabiliser les détections de chaque drone de l'équipe et qui, jugeant que la mission est sur le point de se terminer, va ordonner aux drones de se préparer à quitter leur position.

Il va aussi demander à la station de contrôle de retirer les emplacements de la mission de sa liste des emplacements utilisés.

Il est important de noter que ces instructions ordonnées par le chef d'équipe n'utilisent pas le système de communication implémenté dans le cadre de notre projet (détaillé dans

la partie 5.4). Ces instructions sont passées en appelant directement des méthodes sur les autres drones et sur la station de contrôle, ce qui n'est évidemment pas possible dans la réalité. C'est là un défaut de notre implémentation.

5.1.4 Placement des drones

Lorsque tous les drones ont été choisis par la station de contrôle pour une mission, ils se voient tous attribué un emplacement unique de part et d'autre du passage piéton. C'est une fois que la mission a été lancée que les gardes s'envolent. Dans cette simulation, tous les drones sont à l'origine placés sur des supports de recharge situés sur le toit de plusieurs immeubles de la zone urbaine.

Entre la mise en vol et l'arrivée à cet emplacement unique, nous pouvons compter deux étapes. La première consiste à se rendre à la même position que l'emplacement attribué sur les axes x et z mais à l'altitude de l'immeuble depuis lequel un drone s'est envolé. Cette étape est représentée par un attribut de type booléen ***startFromStation*** valant *true* si le garde en est à cette étape et *false* sinon.

La deuxième étape est de la même manière représentée par l'attribut ***descendToSpot*** de type booléen. Cette étape consiste comme son nom l'indique à faire perdre de l'altitude au drone pour qu'il soit à l'emplacement qui lui a été assigné au démarrage de la mission.

C'est seulement une fois que tous les drones de la mission sont arrivés à destination que le chef ordonne à l'équipe de garde de débuter l'étape suivante, la plus importante : la détection de piétons.

Toutefois, dans la simulation mais également dans la vie réelle, il est possible qu'une voiture ne voie pas à temps les drones et qu'elle empêche l'un de ces derniers d'atteindre son emplacement attribué. Plutôt que de se poser sur la voiture, n'incrémentant donc pas le compteur de drones arrivés à destination et empêchant ainsi la détection de débuter, chaque drone est capable de détecter une voiture en-dessous lors de la descente.

Ainsi, lorsqu'un drone remarque qu'une voiture bloque l'accès à son emplacement, il demande au chef de commencer la mission sans lui. Si la voiture recule suffisamment pour permettre au garde d'accéder à sa destination, ce dernier rejoint ses compagnons pour la détection. Cela dit, si la voiture n'est pas capable de reculer, le drone continuera à survoler cette dernière, ne pouvant pas passer à l'étape de la détection.

Si ce drone bloqué par la voiture est le chef, il envoie une requête à la station de contrôle pour choisir un nouveau chef parmi les drones prêts à détecter des piétons.

Il est intéressant de noter que l'arrivée des drones à destination marque également le début d'émission de lumières grâce à des LEDs intégrées aux drones. Toutes les 0,75 secondes, les gardes vont alterner entre les couleurs bleu et rouge. Cela permet d'alerter les voitures s'approchant du passage piéton de la mission en cours.

5.2 Détection de piétons

5.2.1 Première détection

Il est important que les gardes de traversée soient capables de faire la distinction entre le début et la fin de l'étape de détection. Si les deux se traduisent par une absence de

détection de piétons, l'équipe de drones ne doit pas réagir de la même manière. S'il s'agit d'un début de mission, les gardes doivent laisser le temps à un piéton de s'engager sur le passage. A l'inverse, si l'absence de détection se produit après une détection antérieure de piétons, cela traduit une fin de mission, auquel cas les drones doivent se préparer à prendre vol.

Afin de faire cette distinction, nous avons utilisé un attribut booléen appelé ***pedestrianHasStartedTraversing*** qui est à l'origine *false*.

Ainsi, s'il y a une absence de détection de piétons mais que cet attribut est à *false*, cela signifie que nous sommes au début de la mission et que les gardes peuvent encore attendre.

En revanche, si ***pedestrianHasStartedTraversing*** est à *true* et que l'on ne détecte aucun piéton, cela traduit une fin de mission car tous les piétons ont fini de traverser.

5.2.2 Comptabilisation des détections par le chef d'équipe

Au sein de l'équipe de gardes, seul le chef peut directement communiquer (encore une fois il n'est pas question ici de la communication par broadcast utilisée pour les collisions) avec les autres gardes de la même mission. Ainsi, c'est ce chef qui est responsable de la comptabilisation des détections de chaque drone de l'équipe.

En effet, pour que la mission continue, il faut savoir à chaque itération si au moins l'un des drones de l'équipe a détecté un piéton ou si au contraire aucun des drones n'a pu en détecter un.

Pour effectuer cette comptabilisation, les drones sont dotés d'un attribut appelé ***numberOfDetections*** que seul le chef va utiliser. Au cours de l'étape de détection (représentée par l'attribut booléen ***detection***), chaque drone appelle la méthode ***PedestrianDetectionLaser*** qui va démarrer la détection de piétons via des lancers de rayons (Physics.Raycast sur Unity) et qui, si l'un des rayons détecte un piéton, va passer l'attribut ***pedestrianHasStartedTraversing*** du chef à *true* et va incrémenter le compteur de détections ***numberOfDetections*** du chef d'équipe.

Une fois que tous les drones ont fait appel à cette méthode de détection, l'attribut ***numberOfDetections*** du chef est soit nul, soit strictement positif.

C'est donc à la suite de l'appel à ***PedestrianDetectionLaser*** qu'une autre méthode va être requise : ***ShouldMissionEnd***. Cette dernière ne prend aucun paramètre et renvoie un booléen qui va décider si la mission doit se terminer ou non.

Comme évoqué précédemment, cette décision va se baser sur le fait qu'il s'agisse d'un début ou d'une fin de mission. En l'occurrence, si l'attribut ***PedestrianHasStartedTraversing*** du chef d'équipe est à ***true*** et que son attribut ***numberOfDetections*** vaut 0, la méthode va renvoyer ***true*** car cela signifie que le ou les piétons ont fini la traversée.

C'est seulement une fois cette méthode appelée que l'attribut ***numberOfDetections*** est réinitialisé à 0. De cette manière, à l'itération suivante, le chef d'équipe va à nouveau comptabiliser les détections de ses compagnons avant d'appeler encore une fois ***ShouldMissionEnd***, puis va réinitialiser le compteur de détection à 0, et ainsi de suite.

5.2.3 Timeout

Lorsque les gardes attendent qu'un piéton s'engage au début de l'étape de détection, ils s'autorisent un délai fini correspondant à l'attribut ***droneMissionTimeout*** de la classe ***Parameters***. Cela permet aux drones de lutter contre les faux positifs, qu'il s'agisse de piétons ayant finalement décidé de ne pas traverser ou ayant rapidement traversé avant que l'équipe ne débute la phase de détection. La méthode ***ShouldMissionEnd*** peut effectivement renvoyer *true* lorsque l'attribut ***timerMissionEnd*** atteint ce délai d'expiration, et non pas seulement lorsqu'il n'y a pas de détection.

Cela dit, ce délai de quelques secondes en début de mission n'est actif qu'avant la première détection. Une fois qu'au moins l'un des drones a commencé à détecter une personne (i.e. l'attribut ***pedestrianHasStartedTraversing*** du chef est à true), l'équipe de gardes restera le temps qu'il faudra. De cette manière, les piétons les plus lents comme les personnes âgées ne risquent pas de voir les gardes de traversée s'envoler à cause d'un délai trop long.

Ce même délai d'expiration ***droneMissionTimeout*** est utilisé à la fin de la phase de détection pour prévenir à l'avance les voitures de la fin de mission. Les drones émettent alors une lumière verte avec leurs LEDs.

Que la mission en soit au début ou à la fin, arriver au bout de ce délai d'expiration entraîne un arrêt de la mission représenté par l'attribut ***endOfMission*** valant *true* si la mission prend fin. Les drones prennent alors de l'altitude en même temps jusqu'à être à la même hauteur que leur immeuble respectif.

Ils passent ainsi à l'étape finale représentée par l'attribut booléen ***backToStation*** durant laquelle ils se déplacent jusqu'à leur station de recharge. Une fois arrivés, les gardes deviennent inactifs et voient leurs attributs utiles à la mission tels que ***dronesInMission***, ***chief*** ou encore ***spot*** réinitialisés.

5.3 Autonomie

5.3.1 Évolution du niveau de batterie d'un garde

Si l'équipe de gardes n'est pas censée abandonner un piéton une fois qu'il a commencé à traverser, un drone qui n'a plus suffisamment d'autonomie devra toutefois quitter la mission afin de ne pas se poser sur la route et de ne plus pouvoir se mouvoir.

Dans la simulation, les drones peuvent avoir jusqu'à cinq minutes d'autonomie au maximum. L'autonomie minimale requise pour une mission (en-dessous de laquelle un drone ne peut pas être sélectionné pour la mission) est quant à elle de deux minutes. Cela signifie que si deux minutes après qu'un utilisateur ait appuyé sur le bouton de la borne il n'a toujours pas fini de traverser, un drone qui avait tout juste l'autonomie requise lors du lancement de la mission n'aura plus assez d'autonomie pour achever celle-ci.

Un garde actif – participant donc à une mission – perd une seconde d'autonomie par seconde. Une fois de retour sur son support de charge sur son immeuble d'origine, il est rechargé par un système suffisamment efficace pour offrir deux secondes d'autonomie à un drone, et ce en une seconde.

Si deux minutes semblent suffisantes pour effectuer une mission classique, il est possible qu'il y ait des imprévus. Il se peut aussi qu'un passage piéton soit très emprunté, en particulier à la sortie des écoles.

Dans ce cas, il est envisageable de penser qu'une centaine de personnes traversant le passage de chaque côté puisse amener la mission à durer suffisamment longtemps pour qu'un des drones vienne à manquer d'autonomie.

5.3.2 Requête de remplacement

C'est pour ce genre de cas qu'il a été primordial de mettre en place un système dynamique de remplacement des drones.

En effet, lorsqu'un garde assigné à une mission voit son autonomie passer en-dessous d'un certain seuil basé sur l'autonomie minimale requise pour une mission, nous considérons qu'il n'est plus capable de finir la mission. Dans cette simulation ce seuil est fixée à un tiers de l'autonomie minimale requise pour une mission, ce qui correspond à quarante secondes d'autonomie restantes. Le garde n'a alors que cette durée pour retourner à sa position initiale sur l'immeuble, ce qui est censé être suffisant en sachant que seuls les drones les plus proches du passage sont choisis par la station de contrôle.

Cependant, pour un tel projet dans la vie réelle, il serait intéressant de rendre ce seuil dynamique et dépendant de la distance du drone par rapport au passage piéton, car il est théoriquement possible que tous les drones proches d'un passage soient en-dessous du niveau d'autonomie requis ou bien qu'ils soient déjà sur une autre mission : dans ce cas, un ou plusieurs drones à une distance conséquente pourraient être appelés et ne pourraient pas revenir à leur point de départ en quarante secondes.

Ainsi, une fois en-dessous de ce seuil, le garde envoie une requête de remplacement à la station de contrôle puis se prépare à prendre vol en arborant la couleur noir. La station de contrôle recevant la requête va alors se charger de préparer un nouveau drone avec suffisamment d'autonomie pour effectuer une mission classique.

La station va lui attribuer le même chef et le même emplacement que le drone n'ayant plus assez de batterie. Elle va également avertir le chef d'équipe de ce remplacement : ce dernier va alors retirer le drone défaillant de sa liste et ajouter à la place le nouveau drone.

Ce nouveau drone reprend alors le flambeau de son prédécesseur et réalise l'étape de détection à sa place auprès de l'équipe de gardes.

5.3.3 Changement de chef d'équipe

Le remplacement devient plus complexe lorsque c'est le chef lui-même qui vient à manquer de batterie. Ainsi, quelques secondes avant le remplacement, le chef d'équipe envoie une requête de changement de chef à la station de contrôle. Celle-ci va alors transmettre son statut de chef d'équipe au garde de la mission ayant la plus grande autonomie.

De cette manière l'ancien chef donne toutes les informations qu'il détient, parmi lesquelles nous retrouvons le nombre de détections de piétons comptabilisées ou encore la liste des drones de la mission. Il est prié d'informer tous les drones de ce changement et devient lui-même un garde aux ordres du nouveau chef d'équipe. Quelques secondes plus

tard, à l'instar de n'importe quel drone manquant de batterie, il envoie une requête de remplacement à la station de contrôle.

5.4 Communications entre les drones

L'une des problématiques de notre projet est d'éviter les collisions entre les drones. En effet, lors d'une mission, la distance entre chaque drone est très faible (de l'ordre du mètre) et le risque de collision est alors très élevé ; de plus, des drones appartenant à des missions différentes peuvent se croiser, ce qui entraîne là encore un risque de collision.

Pour éviter cela, et comme les drones sont parfaitement autonomes lorsqu'ils sont en mission, nous avons créé un protocole de communication très minimaliste qui permet aux drones de transmettre des informations utiles aux autres drones, afin d'éviter les collisions.

Dans ce protocole, les messages sont constitués de l'identifiant (nombre entier) du drone expéditeur, et du type du message ; ensuite, en fonction du type, des informations supplémentaires sont fournies. Seuls deux types de messages ont été implémentés : le type *Position* et le type *AntiCollision*.

Le principe de base de ce protocole est que l'envoi des messages se fait en *broascast*, c'est-à-dire qu'un message est envoyé à tous les drones en capacité de le recevoir (en l'occurrence, les drones en capacité de recevoir un message sont ceux situés assez proches de l'expéditeur, la distance maximale étant définie dans la classe ***Parameters***). L'expéditeur envoie les messages sans attendre de réponse. Lorsqu'un drone reçoit un message, il le traite uniquement si ce message le concerne, ce qui dépend du type de message.

Les messages de type *Position* contiennent l'information de position du drone (relative au repère de la scène Unity). Ils sont envoyés à intervalles réguliers. Lorsqu'un drone reçoit une information de position, il la compare avec sa propre position et détermine s'ils sont assez éloignés l'un de l'autre ; si ce n'est pas le cas, alors la procédure « anti-collision » démarre. Dans cette procédure, les seules actions que peuvent réaliser chaque drone sont, soit rester immobile, soit monter à la verticale. C'est le drone avec l'identifiant le plus élevé qui va indiquer à l'autre ce qu'il doit faire, à l'aide des messages de type *AntiCollision*. Ces messages contiennent l'identifiant du drone destinataire ainsi qu'un booléen indiquant si le drone destinataire doit rester immobile ou non. Dans le cas où ce booléen est à vrai, le destinataire reste immobile et l'expéditeur monte à la verticale ; sinon, c'est l'inverse. Lorsque les deux drones sont à nouveau assez éloignés, alors les messages *AntiCollision* arrêtent d'être transmis et les deux drones reprennent leur chemin vers leur destination. Lorsqu'un drone reçoit un message *AntiCollision* mais que son identifiant ne correspond pas à l'identifiant du destinataire, alors le message n'est pas traité.

L'algorithme d'évitement des collisions que nous avons implémenté est détaillé dans l'annexe A.1.

Cette méthode d'évitement des collisions n'est clairement pas idéale, notamment car le seul mouvement possible lors de cette procédure est un mouvement vertical. Or, dans certaines situations, ce comportement ne permettrait pas aux drones d'atteindre leur destination. Par exemple, si l'on a deux drones sur le même axe vertical, et que l'un des deux souhaite aller de haut en bas et l'autre de bas en haut, alors leurs trajectoires vont se confondre ; lorsqu'ils seront tous les deux trop proches, la procédure va faire que les deux

drones vont petit à petit monter, et lorsque le deuxième drone aura atteint sa destination, il ne bougera plus et le premier drone ne pourra pas descendre.
Dans le cadre de notre simulation, cette situation peut se produire lorsqu'un drone manque de batterie et doit se faire remplacer ; dans ce cas, le remplaçant peut se retrouver au-dessus du spot qui lui est alloué avant que le drone remplacé ait fini de remonter. Alors, lorsque les deux seront trop proches, ils vont petit à petit monter jusqu'à ce que le drone remplacé soit arrivé à sa hauteur cible, et celui-ci va à ce moment-là retourner à sa station, ce qui va permettre au drone remplaçant de descendre au spot.

Une autre limitation dans notre implémentation, est que les messages *Position* (et *AntiCollision* lors d'une procédure d'anti-collision) sont envoyés à chaque *frame*, plus précisément dans la méthode ***Update*** appliquée à chaque drone. Cela équivaut à plusieurs dizaines de messages envoyés chaque seconde. Or, dans la réalité, les drones ne pourraient pas envoyer des messages aussi régulièrement, car cela pourrait grandement affecter leur autonomie, et également car ils ne seraient pas en mesure de traiter un aussi grand nombre de messages entrants.

Dans notre implémentation, les collisions sont facilement évitées car l'envoi des messages de position à chaque *frame* permet de démarrer une procédure anti-collision quasiment instantanément après que la distance minimale autorisée entre deux drones ait été franchie ; dans la vie réelle, de par l'envoi moins fréquent de ces messages, la procédure aurait été démarrée plus tardivement, et donc le risque de collision aurait été plus grand. De plus, dans la vie réelle, il est possible que certains messages ne soient pas reçus pour diverses raisons, ce qui peut là aussi retarder le démarrage de cette procédure ; ce n'est pas le cas dans notre simulation.

Enfin, ici, les drones peuvent connaître leur position très précisément, simplement en récupérant leur position relative à la scène dans Unity. Or, dans la réalité, les systèmes de positionnement utilisables dans ce cadre permettraient d'avoir une précision au mieux de quelques centimètres (en utilisant par exemple un GPS RTK), ce qui amènerait une incertitude non négligeable sur le calcul de ces positions. Étant donné que les drones se retrouvent très proches les uns des autres pendant les missions, cette imprécision augmente le risque de collisions. Pour simuler cette contrainte, il aurait fallu ajouter cette incertitude lorsqu'un drone souhaitait récupérer sa position, mais cela n'a pas été implémenté.

6 Tests

Pour tester le bon fonctionnement de notre projet, deux stratégies de tests ont été mises en place.

La première a consisté à réaliser des tests unitaires. Sur Unity, il est possible de réaliser des tests unitaires à l'aide d'un framework dédié : le *Unity Test Framework*. Il permet notamment de reconstruire des éléments d'une scène, et de réaliser des assertions sur l'état ou la position de ces éléments.

Nous avons utilisé ce framework pour tester notre travail sur les collisions, afin de vérifier que les drones parvenaient à s'éviter lorsque leurs trajectoires se croisaient, et que les drones repartaient bel et bien vers leur destination après une manoeuvre d'évitement. Tous ces tests ont été passés avec succès.

Les tests unitaires ont également été utiles pour tester certains éléments d'une mission. Nous avons testé si la station de contrôle affectait le bon nombre de drones à une mission, en fonction du nombre de drones disponibles. Nous avons aussi regardé si la position d'un drone à chaque étape d'une mission était bien celle attendue.

Cependant, les tests unitaires ne permettaient pas de tester toutes les caractéristiques d'une mission ou d'un drone, par exemple lorsqu'il s'agissait de tester le comportement global d'un drone sur l'ensemble d'une mission. De plus, l'utilisation du *Unity Test Framework* nous a posé quelques problèmes de compréhension au début, et même si son utilisation s'est finalement avérée assez simple, il ne permettait pas de tester tous les éléments de la simulation (notamment ceux liés aux éléments graphiques). Pour tester tout cela, la deuxième stratégie a été d'utiliser l'éditeur Unity et de lancer des simulations dans toutes les configurations possibles. Cela permettait d'observer en temps réel le comportement des éléments de la scène, et également de connaître les valeurs de certains attributs intéressants en les affichant dans la console.

7 Projet

7.1 Déroulement du projet

Pour organiser nos *user stories* ainsi que les tâches qui en découlent, nous avons opté pour un Kanban, un outil très utilisé en développement Agile.

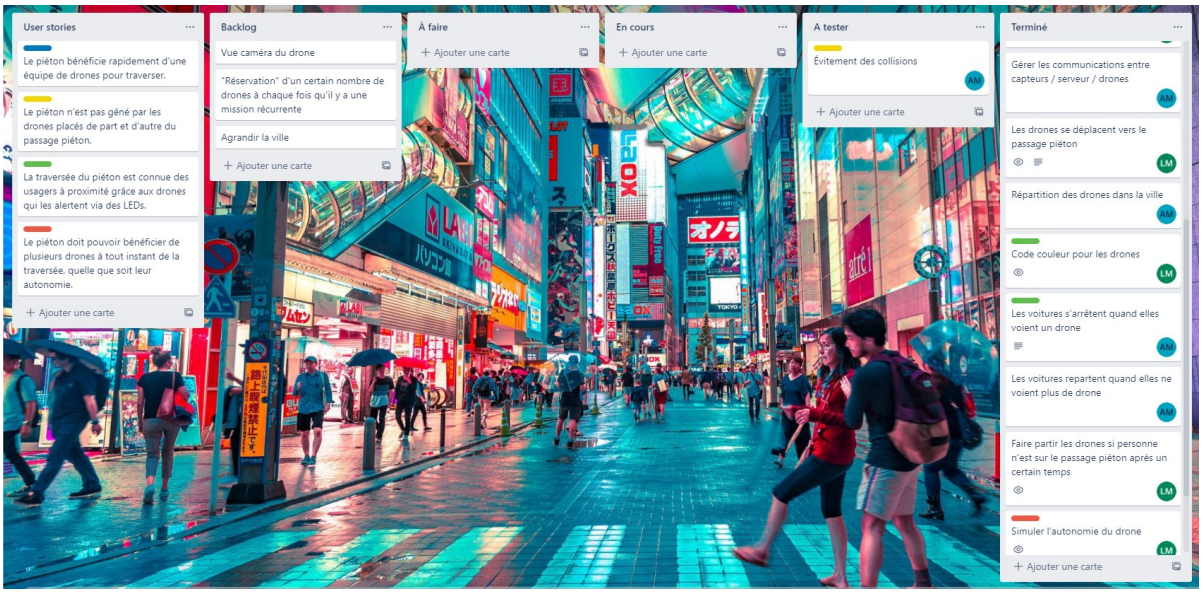

FIGURE 9 – Kanban du projet

A partir des *user stories*, nous avons imaginé une grande quantité de tâches permettant à l'issue du projet de répondre à ces exigences axées sur l'utilisateur, que nous avons placées dans la partie 'Backlog'.

Parmi toutes ces tâches, nous avons choisi celles que nous souhaitions développer, c'est-à-dire la grande majorité des tâches, puis nous les avons placées dans la partie 'À faire'

Puisque nous n'étions que deux dans ce projet, nous ne pouvions pas avancer sur l'intégralité des tâches en même temps ; nous avons donc sélectionné une partie que nous avons placée dans la partie 'En cours', c'est-à-dire en cours de développement.

Avant de placer ces tâches dans la dernière partie 'Terminé', nous les avons placées dans 'À tester'. Cela consistait pour certaines tâches telles que la gestion des collisions où l'arrivée des drones sur un passage à les tester grâce à *Unity Test Framework*.

Pour ce qui est de l'organisation du projet dans le temps, elle correspond assez bien à ce que nous avions prévu au démarrage du projet.

Nous avons validé notre sujet auprès de notre client et encadrant Serge Chaumette au début du mois de février, ce qui nous a laissé environ huit semaines pour développer notre projet.

A l'issue du projet, nous avons établi le diagramme de Gantt effectif suivant :

FIGURE 10 – Diagramme de Gantt effectif du projet

Les grandes tâches sont en gras et chacune d'elles est divisée en sous-tâches. Lorsqu'une case d'une ligne est colorée en vert à une semaine en particulier, cela signifie que nous avons travaillé sur cette sous-tâche spécifique pendant la semaine en question.

Dès le début du projet nous avions décidé de prioriser la modélisation de la ville intelligente et du système de traversée intelligente avant de se lancer dans le développement d'une mission classique des différentes équipes de drones.

Nous avions également prévu de traiter l'autonomie des drones avant les collisions car dans le cadre d'une simulation il nous semblait plus intéressant de développer un système de remplacement dynamique qui pourrait être au moins partiellement repris pour un tel projet dans la vie réelle.

Ce que nous n'avions pas prévu, en revanche, était de passer autant de temps à corriger les différents incidents de missions liés aux couleurs arborées par un drone spécifique et surtout liés aux drones qui sont bloqués par une voiture, et notamment quand ce drone est un chef d'équipe.

Nous aurions probablement pu nous contenter de faire reculer à chaque fois la voiture qui bloque, mais nous avons voulu aller plus loin en prenant en compte le fait que dans la vie réelle il serait courant qu'une voiture s'arrêtant derrière celle qui bloque le drone empêcherait cette dernière de reculer. Nous avons alors fait en sorte qu'un drone continuellement bloqué n'empêche pas la poursuite de la mission, et surtout qu'il permette à celle-ci de se terminer sans lui.

7.2 Résultats

7.2.1 Tâches non implémentées

Dans la partie 'Backlog' du Kanban se trouvent cinq tâches qui n'ont finalement pas été implémentées :

- **'Vue caméra du drone'**
  Cette tâche avait pour but de doter les drones de caméras afin de voir au coin de l'image ce qu'ils voyaient lors de la simulation. Finalement nous avons estimé qu'une telle fonctionnalité n'aurait pas apporté grand chose de plus, et même qu'elle aurait rendu l'affichage peu clair en raison du nombre important de gardes lors d'une mission.
- **'Réservation d'un certain nombre de drones à chaque fois qu'il y a une mission récurrente'**
  Nous pensons toujours que cette fonctionnalité est intéressante et même nécessaire pour un tel projet dans la vie réelle, avec des passages piétons bien plus utilisés que d'autres. En l'occurrence, il nous semble primordial de réserver un nombre fixe de drones à proximité de certaines zones comme des écoles aux heures d'ouverture et de fermeture.
  Cela dit, dans le cadre de la simulation, nous n'avons pas trouvé le temps de simuler des évènements périodiques permettant d'illustrer une telle fonctionnalité. De plus le nombre de drones disponibles est peut-être trop élevé pour visualiser la réservation de certaines drones dans le cas d'un seul individu qui doit manuellement appuyer sur le bouton d'une borne.

— **'Agrandir la ville'**
Si au départ nous souhaitions simuler plusieurs avenues avec un grand croisement complexe, nous avons rapidement rejeté cette idée car elle n'apporterait pas grand chose à la simulation. Pour l'heure il n'y a que trois systèmes de traversée intelligente mais c'est déjà bien assez pour mettre en avant les fonctionnalités de sélection des drones ou bien le fait qu'il est impossible de démarrer une mission s'il n'y a pas suffisamment de drones disponibles.

— **'Vue globale de la ville pour actionner plusieurs systèmes de traversée intelligente'**
Cette fonctionnalité est tout à fait viable et intéressante mais elle s'oppose à la fonctionnalité de la vue à la première personne qui a déjà été implémentée. Ces deux tâches n'ont pas été proposées par la même personne, et ce débat a perduré jusqu'à la fin du projet.
Il s'agit simplement de deux manières différentes de présenter un même système, celui de la traversée intelligente, et c'est pourquoi aucune n'est meilleure que l'autre.

— **'Physique du drone'**
Notre objectif était d'avoir à terme un comportement et des déplacements un peu plus réalistes pour les drones. Nous entendons par là qu'au lieu de les faire se mouvoir avec des méthodes comme ***Vector3.MoveTowards()*** qui se contentent de modifier la position des objets de l'instance, nous voulions faire en sorte qu'un drone produise une portance.
Cette force aurait permis de prendre de l'altitude lorsqu'elle est supérieure à la gravité, d'en perdre lorsqu'elle est inférieure ou bien de se maintenir à une altitude constante si la force de portance est égale au poids du drone.

```
void Move(string vertical="constant")
{
    if(vertical == "constant")
        this.rb.AddForce(Vector3.up * (this.rb.mass * Physics.gravity.magnitude));
    else if(vertical == "up")
        this.rb.AddForce(Vector3.up * (this.rb.mass * 1.5f * Physics.gravity.magnitude));
    else if(vertical == "down")
        this.rb.AddForce(Vector3.up * (this.rb.mass * 0.7f * Physics.gravity.magnitude));
}
```

FIGURE 11 – Tentative d'implémentation de la physique des drones

Lors de la semaine 6 nous avions tenté cette implémentation alternative pour la méthode ***Move*** de Drone.cs avec les arguments *constant*, *up* et *down* servant à choisir la force de portance en fonction de la direction souhaitée.
Cette tentative d'implémentation s'est toutefois avérée peu fructueuse car dans de nombreux cas les drones ne parvenaient pas à contrer la gravité, ou alors l'un des gardes d'une mission s'écrasait sur la route alors que les autres arrivaient plus ou moins à se maintenir à l'altitude de leur emplacement attribué.
N'arrivant pas à trouver une implémentation satisfaisante, nous avons remis cette tâche à plus tard pour nous concentrer sur d'autres tâches plus importantes selon nous telles que la gestion des collisions et la gestion de l'autonomie des drones.
Finalement, nous avons estimé qu'il y avait peu d'intérêt à simuler cet aspect de

portance pour ce projet, car c'est une problématique qui a déjà été pensée maintes et maintes fois pour les drones autonomes et semi-autonomes.
Ainsi, dans l'optique d'un passage de la simulation à un projet dans la vie réelle, il nous paraissait plus pertinent de travailler des aspects plus spécifiques à ce système de traversée intelligente tels que les remplacements de drones et l'évitement des collisions.

7.2.2 De la simulation à la vie réelle

Cette implémentation du projet étant une simulation sur Unity, elle n'est par définition pas une représentation fidèle et réaliste d'un tel projet dans la vie réelle.

Par exemple, nous avons utilisé pour la détection des piétons des lancers de rayons, fonctionnalité phare d'Unity.

Dans la vie réelle, nous pourrions imaginer utiliser des capteurs aux caractéristiques similaires tels que des capteurs LiDAR (laser imaging detection and ranging), capables de détecter des objets et d'estimer leur distance grâce à des lasers. Si ce type de capteur pourrait sans doute fonctionner, il nous semblerait plus judicieux de faire appel à des caméras embarquées à l'avant de chaque drone et capables d'identifier un être humain grâce à des algorithmes de reconnaissance faciale qui existent déjà.

Un autre point qui différerait sûrement par rapport à un projet dans la vie réelle est l'emplacement des drones.

Si dans la simulation ils se trouvent sur les immeubles de la ville, il y aurait sans doute des problèmes de réglementation en réalité. Il serait peut-être plus réaliste de stocker quelques drones dans des stations au sol inaccessibles de l'extérieur et pouvant s'ouvrir afin de laisser les drones sélectionnés pour une mission prendre vol.

Si globalement notre code est rempli de spécificités propres à Unity, nous pensons tout de même que de nombreux aspects peuvent être transposés dans un projet d'envergure réelle.

Par exemple, le principe du chef d'équipe qui a la responsabilité de coordonner les autres gardes de la mission et de comptabiliser les détections de l'équipe peut être facilement repris et réécrit dans un langage de programmation adapté ou même en pseudo-code, bien que ces détections pourraient changer de forme comme évoqué précédemment.

Nous pouvons évoquer d'autres exemples tels que le système de remplacement dynamique de drones en cas de manque d'autonomie grâce à des requêtes, ou la gestion des collisions via des émissions de messages en broadcast par les drones actifs.

Le système lui-même et notamment ses bornes pourraient être adaptés à la vie réelle. Pour limiter les faux positifs, il serait envisageable de réserver ce système à des gens ayant un compte (gratuit) pour le service et de remplacer le bouton par un détecteur de badge ou de QR code. Nous pourrions même imaginer un système de score qui déterminerait la priorité de l'utilisateur.

Par exemple, le fait d'actionner le système pour au final ne pas traverser (et donc laisser l'essaim bloquer le passage aux voitures pendant plusieurs secondes) ferait baisser le score, tandis qu'un fonctionnement normal de la mission avec des drones qui détectent à temps au moins un piéton pour finalement terminer la mission en quelques dizaines de

secondes augmenterait le score de l'utilisateur ayant présenté son badge ou son téléphone à la borne.

Baisser le score dans le cas d'un nombre de remplacements trop important de drones dû à un manque de batterie ne serait cependant pas très juste, surtout dans le cas de passages très empruntés où une personne activant le système pousserait des dizaines d'autres personnes à traverser, augmentant ainsi le temps de la mission. Cela dissuaderait sans doute les utilisateurs d'appeler une équipe de drones, ce qui serait contre-productif. Il serait plus pertinent de garder le score constant au lieu de l'augmenter comme dans le cas d'un fonctionnement normal et court.

Ce score pourrait influer sur le temps de réponse des bornes, sur la sélection des drones (les drones les plus proches seraient sélectionnés dans le cas de scores moyens ou bons alors que dans le cas de scores mauvais certains drones plus éloignés seraient choisis) ou même sur l'accessibilité du système de traversée intelligente : un score catastrophique traduisant une volonté de provoquer des faux positifs en activant des bornes très régulièrement et sans aucune raison entraînerait une perte totale de l'accès au système.

8 Conclusion

La particularité de ce projet par rapport ceux de l'UE PFE est que nous avons nous-mêmes imaginé le sujet.

Nous avons alors dû faire face à un compromis : notre projet devait être à la fois innovant mais également réaliste.

C'est en réfléchissant au futur des villes, qu'on appelle la ville intelligente ou ville connectée, que nous avons trouvé l'inspiration de ce projet.

Le système de traversée intelligente que nous avons imaginé et simulé répond selon nous à ces critères de réalisme et d'innovation. Il fait intervenir des drones autonomes ne nécessitant aucune assistance d'êtres humains, ce qui peut être perçu comme futuriste, mais ce système interagit de manière crédible et vraisemblable avec les éléments d'une ville et ses usagers.

Activable via des bornes placées à des passages piétons stratégiques, le système est capable d'alerter les voitures à proximité de la traversée prochaine d'un piéton et de dissuader ces mêmes voitures de forcer le passage.

De cette manière les piétons ont moins d'hésitation pour s'engager et peuvent effectuer leur traversée plus sereinement.

Les drones étant suffisamment visibles, les voitures peuvent adapter leur vitesse en ralentissant afin optimiser leur temps d'attente devant les drones.

Bien que nous n'ayons pas implémenté l'intégralité des éléments de la simulation que nous avions prévus à l'origine, nous pensons que les fonctionnalités principales permettant de mettre en avant toutes les spécificités de notre système ont été correctement développées.

Le fait que les gardes soient capables d'éviter les collisions au sein d'une même mission ou même entre plusieurs essaims différents ainsi que la gestion de l'autonomie, qui dans

cette simulation est loin d'être infinie, permettent de visualiser une partie conséquente des problématiques auxquelles un tel projet devrait répondre dans la vie réelle.

A cela, nous pouvons ajouter les différentes manières d'effectuer une même mission, avec un piéton qui peut décider d'utiliser ou non le système, auquel cas les drones assignés à la mission devront réagir différemment.

Toute cette diversité d'incidents qui vient s'ajouter à une simple détection de piéton apporte selon nous un réel crédit à la simulation.

Références

[1] Serge Chaumette. Resilience by design is mandatory to support the certification of the embedded/artificial intelligence of an autonomous swarm of drones. In 2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC), pages 1–6, 2020.

[2] Jean-Aimé Maxa. Architecture de communication sécurisée d'une flotte de drones. PhD thesis, Université Toulouse 3 Paul Sabatier, 2017.

[3] Sabrina Islam Muna, Srijita Mukherjee, Kamesh Namuduri, Marc Compere, Mustafa Ilhan Akbas, Péter Molnár, and Ravichandran Subramanian. Air corridors : Concept, design, simulation, and rules of engagement. Sensors, 21(22), 2021.

A Annexe

A.1 Algorithme d'évitement des collisions

Un pseudo-code de l'algorithme est donné en page suivante.

Cet algorithme doit fonctionner si deux drones sont situés trop proches les uns des autres, mais aussi s'il y a plus de deux drones trop proches les uns des autres. C'est pour cela que nous avons rajouté un attribut aux drones : `tooCloseDrones`. Il s'agit d'un dictionnaire qui contient l'ensemble des identifiants des drones pour lesquels une procédure d'évitement est en cours, et qui leur associe un booléen correspondant à la *stopInstruction* : si ce booléen vaut *true*, alors c'est le drone correspondant qui effectue la manoeuvre d'évitement.

Un deuxième attribut a été rajouté : `antiCollisionTarget`. Celui-ci correspond à la position cible temporaire du drone, vers laquelle il doit se diriger pour éviter la collision. Si le dictionnaire `tooCloseDrones` contient au moins un *true*, alors le drone doit rester immobile (donc `antiCollisionTarget` correspond à la position actuelle du drone) ; sinon, comme indiqué précédemment, le drone doit se déplacer vers le haut (donc `antiCollisionTarget` correspond à une position située au-dessus du drone).

La procédure d'évitement doit être enclenchée dès lors que le drone reçoit un message de position située trop proche de lui (nous avons défini une valeur minimale, c'est la variable `minimumDistanceAllowed`), c'est pour cela que tout se passe dans la méthode `IncomingMessage(msg)` qui est appelée dès qu'un message est reçu.

Considérons les drones X et Y.
Si le message reçu par X est de type *Position*, et que la position reçue est trop proche de celle de X, alors la procédure commence. X prend en main la procédure uniquement dans le cas où son identifiant est supérieur à celui de Y. Dans ce cas, si X est situé plus haut que Y, alors c'est X qui va devoir se déplacer vers le haut et Y rester immobile, sinon c'est l'inverse ; X envoie alors à Y un message *AntiCollision* lui indiquant cette information (ce qu'on appelle la *stopInstruction*). En même temps, X a mis à jour les attributs `tooCloseDrones` et `antiCollisionTarget` en fonction de toutes ces données.
Si le message reçu par X est de type *AntiCollision*, et qu'il est bien le destinataire du message, alors il met à jour les deux attributs en fonction de la *stopInstruction* reçue.

```
# attributs du drone
Dictionnary<int, bool> tooCloseDrones
Position antiCollisionTarget


# methode du drone pour traiter un message entrant
def IncomingMessage(Message msg):

    if msg.type == Position:

        if Distance(this.position, msg.position) < minimumDistanceAllowed:

            if not msg.droneId in tooCloseDrones.keys:
                tooCloseDrones.Add(msg.droneId, false)

            antiCollisionTarget = this.position

            if this.id > msg.droneId:

                if this.position.y < msg.position.y:
                    BroadcastMessage(type=AntiCollision, droneId=this.id, targetId=msg.droneId, stopInstruction=false)
                    tooCloseDrones[msg.droneId] = true
                else:
                    BroadcastMessage(type=AntiCollision, droneId=this.id, targetId=msg.droneId, stopInstruction=true)
                    if not true in tooCloseDrones.values:
                        antiCollisionTarget.y += 1

        else:
            del tooCloseDrones[msg.droneId]

    else if msg.type == AntiCollision:
        if this.id != msg.targetId or this.id > msg.droneId: return

        if not msg.droneId in tooCloseDrones.keys:
            tooCloseDrones.Add(msg.droneId, true)

        antiCollisionTarget = this.position

        if not msg.stopInstruction:
            tooCloseDrones[msg.droneId] = false
            if not true in tooCloseDrones.values:
                antiCollisionTarget.y += 1

# methode du drone pour se deplacer (appelee a chaque frame)
def Move():
    if len(tooCloseDrones) == 0:
        MoveTo(initialTarget)
    else:
        MoveTo(antiCollisionTarget)
    BroadcastPosition()
```