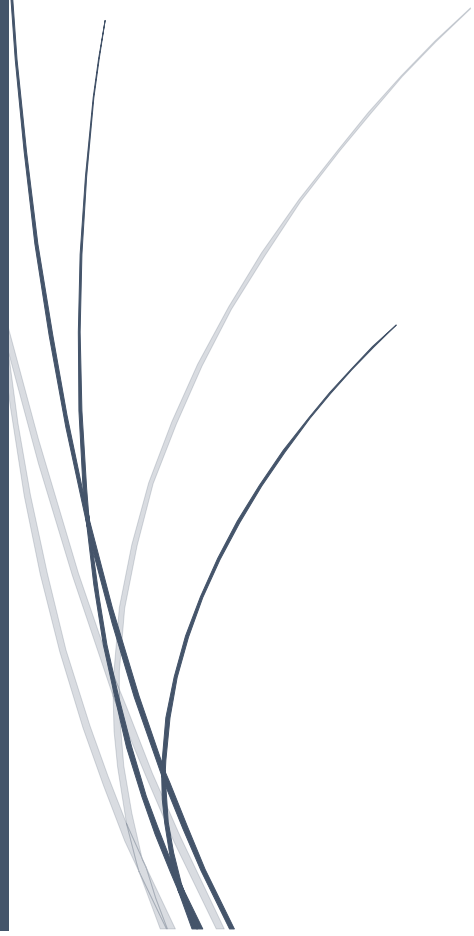
M2 DAS

Rapport de projet

RT0911 – Test et vérification



Thibaut Allart
UNIVERSITE DE REIMS CHAMPAGNE-ARDENNES

Dans ce rapport, je vais expliquer comment j'ai conçu mon application, pourquoi j'ai fait certains choix plutôt que d'autres, mais aussi montrer des exemples d'exécution. Le code du projet est situé dans le dépôt GitHub suivant : https://github.com/Faige-FR/RT0911-Thibaut-ALLART . Ce dépôt ne contient qu'une version avec les fichiers nécessaires et est publique.

Le projet a initialement été réalisé dans un dépôt privé, contenant les projets de toutes les autres matières, ainsi que des fichiers que j'ai créé au début et qui ne s'avèrent finalement pas utiles pour le rendu. J'ai donc décidé de ne publier que le contenu intéressant, et non l'intégralité de mon travail.

Le code contient des commentaires afin d'en faciliter la lecture et pour résoudre les problèmes rencontrés plus facilement. L'ensemble du projet a été testé avec une version Python 3.11.5.

Algorithmes

Au début du projet, je pensais pouvoir séparer quelques étapes de l'algorithme principal dans différents fichiers, sans forcément créer de classes pour faciliter la rédaction du code. Au fur et à mesure que le projet avançait, je me suis rendu compte de mon erreur mais j'ai décidé de continuer plutôt que de reprendre plusieurs heures pendant les cours pour refaire un projet complet.

J'ai donc au total trois fichiers : le fichier principal qui exécute le programme, un fichier qui contient une fonction initialisant les données de mon véhicule, ainsi qu'un fichier qui gère la connexion et la réception des messages du broker MQTT. En outre, j'ai réalisé des scripts qui me permettent de tester les fonctionnalités de l'application seul, à partir du moment où le broker restait fonctionnel. Seule la partie sur la collision peut être erronée, puisque malgré les tests effectués en ajoutant de nouveaux véhicules, je ne recevais aucun message dans la file « positions ».

Le fichier « mqttClient.py » permet de gérer la connexion avec le broker MQTT. Le client peut ainsi se connecter à chaque file donnée en cours et recevoir les messages envoyés sur chacune d'entre elles. J'avais commencé à utiliser des variables globales, même si c'est plutôt déconseillé et peu « propre », pour me faciliter la tâche dans un premier temps. J'aurais pu créer une classe « singleton » à laquelle j'aurais pu donner des attributs en fonction des messages reçus. J'ai donc créé quatre variables globales, qui me permettent de savoir dans la fonction principale si le véhicule peut démarrer, si l'uppertester a envoyé une demande au véhicule, une troisième qui enregistre le statut de chaque feu de circulation et une dernière qui enregistre la position des véhicules en état de marche. Même si des variables globales sont utilisées, elles ne restent modifiées que par un seul acteur : le client MQTT (lorsque le broker envoie une nouvelle information) ; cela me permet d'être sûr que mes variables ne seront pas erronées lors du traitement. La seule exception reste la variable indiquant si l'uppertester a une requête, puisque le statut de celle-ci repasse à l'état initial lorsque la réponse a été envoyée avec succès.

Le fichier d'initialisation des données du véhicule « car.py » va simplement lire le contenu d'un fichier « car.ini », qui contient l'identifiant du véhicule, son type, ses positions de départ et d'arrivée, la direction lors du premier déplacement, sa vitesse maximale et l'ensemble des tronçons qui seront utilisés par le véhicule pour se déplacer sur la carte.

Le fichier « main.py » contient l'algorithme principal de déplacement du véhicule. Le code peut paraître complexe mais ne l'est pas vraiment en réalité, puisque la plupart des calculs prend en compte les différentes directions possibles en fonction de l'état des feux de circulation et des autres véhicules de la carte. Au départ, on récupère les informations du véhicule et de la carte, notamment la position

des feux de circulation. Ensuite, on se connecte au broker MQTT afin de pouvoir communiquer avec lui. Une fois le signal de départ réceptionné, le véhicule démarre. A chaque tour, on envoie notre position au broker pour que chaque véhicule soit averti de notre présence. On regarde si notre position correspond à celle du point de destination. Si ce n'est pas le cas, on recherche le tronçon à emprunter ainsi que la position du prochain feu s'il y en a un sur ce tronçon. Ensuite, on vérifie qu'aucun véhicule ne se dirige dans la même direction que nous et que l'on n'entrera pas en collision avec lui. Une distance maximale parcourable est alors établie. Par défaut, on peut parcourir 999 cases ; cela signifie qu'aucun véhicule ne risque de se retrouver sur notre chemin. Une fois les collisions évitées, on s'assure de ne pas traverser un feu au rouge. On choisira enfin la distance minimale parmi la vitesse du véhicule, la distance avant d'arriver au prochain changement de direction (dans la plupart des cas, un feu y est situé) et la distance avec le véhicule le plus proche s'il est devant nous et dans la même direction. On enverra cette distance minimale au broker pour que chaque véhicule connaisse notre nouvelle position.

Les fichiers « generateLights.py », « go.py » et « senderRequestUT.py » permettent respectivement de générer un état aléatoire pour chaque feu dans chaque direction, d'envoyer un signal « GO » dans la file « top » et d'envoyer une requête de la part de l'uppertester au véhicule avec l'identifiant « 3 ». Ces fichiers m'ont servi pour réaliser les tests de l'application. L'uppertester devraient être capable d'envoyer différentes requêtes au client. Etant donné que nous n'avons finalement qu'un type de message à traiter, je n'ai pas ajouté les fonctions supplémentaires que j'avais commencé à concevoir pour lui répondre. J'avais prévu de traiter les cas suivants : envoi de la position, envoi de l'identifiant du tronçon sur lequel le véhicule se situe, envoi de l'identifiant d'un véhicule devant nous, envoi de l'horloge système et envoi de la vitesse du véhicule. Chaque fonction aurait également envoyé le temps entre la réception de la demande et l'envoi du message. Pour les cas où la fonction demandée est inconnue par le client, une fonction par défaut aurait renvoyé « -1 » à l'uppertester, en plus du temps de réponse et de l'identifiant de mon véhicule.

Des détails supplémentaires sur certains aspects du programme sont indiqués en commentaire dans le code. Pour faciliter l'exécution, le véhicule ne se dirige pas sur les bords de la carte pour retourner de l'autre côté : le véhicule ne se rend pas à la position X ou Y égale à 100 pour éviter de gérer les cas avec des modulos. De plus, de l'optimisation aurait sûrement pu être ajoutée à certains endroits pour réduire le temps d'exécution du programme.

Utilisation de la file MQTT et échanges

Les communications avec le broker MQTT assurent que le véhicule peut se déplacer sans risque. Le client MQTT récupère les informations pour que la boucle principale du programme puisse les traiter. Depuis cette dernière, on enverra directement les informations vers la file appropriée au besoin.

Les différents messages envoyés concernent les files « vehicle » et « RESP ». Pour les messages dans « vehicle », on envoie l'identifiant du véhicule (« 3 » dans mon cas), son type (toujours « 1 »), ses positions en x et en y (toujours entre 0 et 99), sa direction (entre 0 et 4, 4 étant pour moi l'état « arrivé ») et la vitesse du véhicule (nombre entier). Un exemple de message valide est par exemple :

```
{"id": 3, "vtype": 1, "x": 50, "y": 18, "dir": 2, "speed": 3}
```

Pour la file « RESP », on enverra toujours l'identifiant de notre véhicule, son horloge interne (sous forme de timestamp UNIX) et sa position. Un message valide pourrait donc être :

```
{"id": 3, "temps": 1715166587, "position": "36,42"}
```

En ce qui concerne la réception des messages, tout est effectué par le client MQTT pour que la boucle principale puisse réaliser correctement les déplacements du véhicule. Pour cela, on récupère simplement les informations que l'on décode et formate en JSON afin de les utiliser facilement. Le programme principal va alors utiliser la position des véhicules (file « positions ») et l'état des feux de circulation (file « lights ») pour s'adapter à chaque situation. Ces messages sont directement convertis au format JSON pour pouvoir les traiter de la même manière que l'ensemble des messages échangés dans ce projet. Il existe un message générique « GO » dans la file « top » qui indique à chaque véhicule qu'ils peuvent démarrer. Avant d'envoyer un message dans la file « RESP », il faut au préalable recevoir un message dans la file « UT » qui indiquera l'identifiant du véhicule à qui l'uppertester adresse une requête.

Exécution du programme

Pour démarrer l'application, il suffit de lancer la commande « python main.py » dans un terminal (remplacer « python » par « python3 » selon le mode d'installation). Il est nécessaire d'envoyer un signal « GO » dans la file « top » pour que le véhicule démarre et d'avoir la liste des feux de circulation et de leur état dans la file « lights ».

L'exemple suivant illustre l'envoi et la réception des messages durant le trajet du véhicule. Seule la file « positions » n'apparaît pas car je ne recevais pas de message supplémentaire durant mes tests avec plusieurs véhicules.

Exécution du programme :

```
$ python main.py

Lights :  [(1, 20, 10), (2, 40, 10), (3, 60, 10), (4, 90, 10), (5, 20, 20), (6, 40, 20), (7, 90, 20), (8, 40, 30), (9, 90, 40), (10, 10, 40), (11, 20, 40), (12, 40, 40), (13, 70, 40), (14, 40, 50), (15, 70, 50), (16, 90, 50), (17, 30, 70), (18, 40, 70), (19, 60, 70), (20, 70, 70), (21, 90, 70), (22, 10, 80), (23, 30, 80), (24, 60, 80), (25, 70, 80), (26, 90, 80), (27, 10, 90), (28, 70, 90), (29, 90, 90)]

Connected to MQTT brokers

Starting at: [10, 10]

Ending at: [70, 84]


Received message on TOPIC 'lights': {"1": "1,1,0,0", "2": "1,1,1,0", "3": "1,0,0,0", "4": "0,0,0,0", "5": "1,0,1,1", "6": "1,1,1,1", "7": "0,0,1,1", "8": "0,0,1,0", "9": "1,0,1,1", "10": "0,1,0,1", "11": "1,0,1,1", "12": "0,0,0,0", "13": "1,0,0,0", "14": "1,1,1,0", "15": "1,1,0,1", "16": "1,0,0,0", "17": "0,0,0,0", "18": "1,1,0,1", "19": "1,0,0,0", "20": "0,0,0,1", "21": "1,0,1,0", "22": "1,1,0,0", "23": "0,1,1,0", "24": "0,1,1,1", "25": "0,1,0,0", "26": "1,0,0,1", "27": "1,0,0,0", "28": "0,0,1,0", "29": "0,0,1,1"}

Received message on TOPIC 'lights': {"1": "0,0,1,0", "2": "1,1,0,0", "3": "1,0,1,0", "4": "0,0,0,0", "5": "0,0,0,1", "6": "1,0,1,0", "7": "1,1,0,0", "8": "0,1,1,1", "9": "0,0,1,1", "10": "0,1,1,0", "11": "0,0,1,0", "12": "0,0,0,1", "13": "0,1,1,0", "14": "0,0,1,0", "15": "1,0,0,1", "16": "0,1,1,1", "17": "0,0,1,0", "18": "1,1,0,1", "19": "1,0,1,1", "20":
```

"1,1,0,0", "21": "0,1,0,1", "22": "0,1,0,1", "23": "1,0,0,1", "24": "1,1,1,1", "25": "1,1,1,0", "26": "1,0,1,0", "27": "0,0,1,0", "28": "0,1,0,1", "29": "1,0,1,0"}

Received message on TOPIC 'top': GO

=== DEPART ===

Sending: {"id": "3", "vtype": 1, "x": 10, "y": 10, "dir": 0, "speed": 3} on 'vehicle'

Received message on TOPIC 'vehicle': {"id": "3", "vtype": 1, "x": 10, "y": 10, "dir": 0, "speed": 3}

Received message on TOPIC 'lights': {"1": "0,1,0,0", "2": "1,1,1,0", "3": "1,1,0,1", "4": "1,0,0,1", "5": "1,1,1,0", "6": "0,1,1,0", "7": "0,0,1,0", "8": "1,1,0,1", "9": "1,0,0,0", "10": "1,0,0,1", "11": "1,0,1,1", "12": "0,1,1,0", "13": "1,1,1,1", "14": "1,1,1,0", "15": "1,1,0,0", "16": "1,0,1,1", "17": "1,1,0,1", "18": "1,1,1,0", "19": "0,0,0,1", "20": "1,1,1,1", "21": "1,1,1,0", "22": "0,0,0,1", "23": "0,0,1,1", "24": "0,1,0,0", "25": "1,0,0,0", "26": "0,1,0,0", "27": "1,0,0,1", "28": "0,1,1,0", "29": "1,1,1,0"}

Sending: {"id": "3", "vtype": 1, "x": 13, "y": 10, "dir": 0, "speed": 3} on 'vehicle'

Received message on TOPIC 'vehicle': {"id": "3", "vtype": 1, "x": 13, "y": 10, "dir": 0, "speed": 3}

Sending: {"id": "3", "vtype": 1, "x": 16, "y": 10, "dir": 0, "speed": 3} on 'vehicle'

Received message on TOPIC 'vehicle': {"id": "3", "vtype": 1, "x": 16, "y": 10, "dir": 0, "speed": 3}

Sending: {"id": "3", "vtype": 1, "x": 19, "y": 10, "dir": 0, "speed": 3} on 'vehicle'

Received message on TOPIC 'vehicle': {"id": "3", "vtype": 1, "x": 19, "y": 10, "dir": 0, "speed": 3}

Received message on TOPIC 'lights': {"1": "0,1,1,0", "2": "1,0,0,0", "3": "0,0,1,1", "4": "0,1,0,1", "5": "1,0,1,1", "6": "1,0,0,0", "7": "0,0,0,1", "8": "0,0,1,1", "9": "1,0,1,0", "10": "0,0,1,1", "11": "1,0,0,0", "12": "0,1,1,1", "13": "1,1,0,0", "14": "0,1,0,1", "15": "0,0,1,0", "16": "0,0,0,1", "17": "0,0,1,0", "18": "0,1,0,0", "19": "0,0,1,0", "20": "0,1,0,0", "21": "1,0,1,0", "22": "1,1,1,0", "23": "1,0,1,1", "24": "0,0,1,0", "25": "1,1,1,0", "26": "0,1,1,0", "27": "0,1,0,1", "28": "1,0,0,1", "29": "0,1,1,1"}

Sending: {"id": "3", "vtype": 1, "x": 20, "y": 10, "dir": 0, "speed": 3} on 'vehicle'

Received message on TOPIC 'vehicle': {"id": "3", "vtype": 1, "x": 20, "y": 10, "dir": 0, "speed": 3}

Received message on TOPIC 'UT': {"id": 3}

Uppertester's request captured. Sending: {"id": "3", "temps": 1715174699, "position": "23,10"} on 'RESP'

Sending: {"id": "3", "vtype": 1, "x": 23, "y": 10, "dir": 0, "speed": 3} on 'vehicle'

Received message on TOPIC 'vehicle': {"id": "3", "vtype": 1, "x": 23, "y": 10, "dir": 0, "speed": 3}

…

A la fin du programme, on obtient le résultat suivant.

Fin du programme :

Received message on TOPIC 'lights': {"1": "1,0,1,0", "2": "0,0,0,1", "3": "1,1,0,0", "4": "0,0,0,1", "5": "1,1,1,1", "6": "0,1,1,1", "7": "0,1,1,1", "8": "0,1,1,1", "9": "1,0,1,0", "10": "0,0,1,1", "11": "0,0,0,0", "12": "1,1,0,1", "13": "1,1,1,0", "14": "0,1,1,0", "15": "1,0,1,0", "16": "1,1,1,1", "17": "1,0,1,1", "18": "0,1,1,1", "19": "1,0,1,0", "20": "0,0,1,1", "21": "1,0,1,0", "22": "0,0,1,1", "23": "1,1,1,1", "24": "1,1,0,1", "25": "0,0,0,0", "26": "1,1,0,0", "27": "1,1,0,0", "28": "1,1,0,1", "29": "1,0,1,0"}

Sending: {"id": "3", "vtype": 1, "x": 70, "y": 80, "dir": 0, "speed": 3} on 'vehicle'

Received message on TOPIC 'vehicle': {"id": "3", "vtype": 1, "x": 70, "y": 80, "dir": 0, "speed": 3}

Sending: {"id": "3", "vtype": 1, "x": 70, "y": 83, "dir": 1, "speed": 3} on 'vehicle'

Received message on TOPIC 'vehicle': {"id": "3", "vtype": 1, "x": 70, "y": 83, "dir": 1, "speed": 3}

Sending: {"id": "3", "vtype": 1, "x": 70, "y": 84, "dir": 1, "speed": 3} on 'vehicle'

Received message on TOPIC 'vehicle': {"id": "3", "vtype": 1, "x": 70, "y": 84, "dir": 1, "speed": 3}

Received message on TOPIC 'lights': {"1": "1,1,1,0", "2": "1,0,0,0", "3": "0,1,0,0", "4": "1,1,0,0", "5": "0,1,1,1", "6": "1,1,1,0", "7": "0,1,0,0", "8": "0,0,1,0", "9": "0,1,0,1", "10": "0,1,1,1", "11": "0,1,0,1", "12": "1,1,0,0", "13": "0,0,1,1", "14": "0,0,1,1", "15": "0,1,1,0", "16": "0,0,1,1", "17": "1,1,0,1", "18": "0,0,0,0", "19": "1,0,0,1", "20": "0,1,0,0", "21": "0,0,0,1", "22": "1,1,1,0", "23": "0,1,0,1", "24": "0,0,1,0", "25": "1,1,1,1", "26": "0,0,0,0", "27": "0,1,0,0", "28": "1,0,0,0", "29": "0,0,1,0"}

=== ARRIVEE ===

Received message on TOPIC 'lights': {"1": "0,0,0,1", "2": "1,1,0,1", "3": "1,1,0,1", "4": "0,0,0,0", "5": "0,0,1,0", "6": "0,1,1,0", "7": "1,0,0,1", "8": "1,1,1,0", "9": "1,1,0,1", "10": "0,0,0,0", "11": "1,1,1,1", "12": "1,1,0,0", "13": "0,1,1,1", "14": "0,1,0,1", "15": "0,0,1,1", "16": "0,0,0,0", "17": "1,1,1,0", "18": "1,0,1,0", "19": "1,1,0,1", "20": "0,0,1,0", "21": "0,1,0,0", "22": "0,0,1,0", "23": "0,1,1,1", "24": "1,0,1,0", "25": "1,1,1,1", "26": "1,0,1,1", "27": "1,0,1,1", "28": "1,1,0,0", "29": "0,0,0,1"}

Received message on TOPIC 'lights': {"1": "0,0,1,0", "2": "1,0,0,1", "3": "0,1,0,1", "4": "1,1,1,0", "5": "1,1,0,0", "6": "1,1,0,1", "7": "0,0,1,1", "8": "1,0,0,0", "9": "0,0,0,1", "10": "1,1,1,1", "11": "0,1,0,0", "12": "0,1,1,0", "13": "1,0,0,1", "14": "0,1,1,0", "15": "0,1,1,0", "16": "0,1,1,1", "17": "1,0,0,1", "18": "0,1,1,1", "19": "1,0,0,1", "20": "0,1,0,0", "21": "0,1,1,0", "22": "0,1,1,1", "23": "1,1,0,1", "24": "1,0,0,1", "25": "1,0,0,0", "26": "1,1,1,1", "27": "0,1,1,0", "28": "1,1,1,1", "29": "1,1,0,0"}

Received message on TOPIC 'lights': {"1": "0,1,1,1", "2": "1,0,1,0", "3": "1,0,1,0", "4": "1,0,1,0", "5": "0,0,0,0", "6": "1,1,1,1", "7": "1,0,1,1", "8": "0,0,0,0", "9": "1,1,0,0", "10": "0,1,1,1", "11": "1,1,0,0", "12": "1,0,0,0", "13": "0,1,0,0", "14": "0,1,1,1", "15": "0,0,0,0", "16": "0,1,0,1", "17": "0,1,1,0", "18": "1,1,0,0", "19": "1,0,0,1", "20": "0,1,1,0", "21": "1,1,0,1", "22": "1,1,0,0", "23": "1,0,1,1", "24": "1,0,0,1", "25": "0,0,1,0", "26": "1,1,1,0", "27": "1,1,1,0", "28": "1,0,1,1", "29": "0,0,1,0"}

Received message on TOPIC 'lights': {"1": "0,1,1,0", "2": "0,0,0,0", "3": "0,1,0,1", "4": "0,1,0,0", "5": "0,1,0,0", "6": "0,0,0,0", "7": "1,0,1,1", "8": "1,1,1,0", "9": "0,1,1,0", "10": "1,1,0,0", "11": "1,0,1,0", "12": "1,0,1,0", "13": "0,0,0,1", "14": "0,0,1,0", "15": "1,1,0,0", "16": "0,1,1,0", "17": "1,0,0,0", "18": "0,1,1,1", "19": "1,1,0,0", "20": "1,1,1,0", "21": "0,0,0,0", "22": "0,1,0,0", "23": "0,0,0,0", "24": "0,1,0,1", "25": "1,0,0,1", "26": "0,0,0,1", "27": "1,0,1,1", "28": "0,1,0,1", "29": "1,1,1,1"}