Projet de programmation C# - Gestion d'un carrefour

1. Objectif

L'application programmée permet de gérer un carrefour de feux tricolores.

Le carrefour est composé de:

- Quatre feux tricolores
- Quatre appuis de bouton poussoir pour les piétons
- Quatre détecteurs de véhicules

La commande des couleurs des feux respecte l'ordre d'affichage en France. Le paramétrage des temps d'affichage des feux vert et rouge (possibilité d'ajouter les feux oranges en modifiant légèrement le programme) se fera depuis un fichier texte de configuration qui aura cette forme:

```
Mini Maxi
60 150
45 300
```

Les valeurs données correspondent respectivement:

- Temps minimum au vert sur voie 1
- Temps maximum au vert sur voie 1
- Temps minimum au vert sur voie 2
- Temps maximum au vert sur voie 2

La voie 1 est composée des feux numérotés 1 et 2, tandis que la voie 2 comporte les feux 3 et 4.

L'application permet l'archivage du fonctionnement du programme. Cet historique ne comportera que les événements du carrefour, à savoir les appuis sur les bouton-poussoirs des piétons et les détecteurs de véhicules.

Le fichier historique sera au format CSV, avec une tabulation comme séparateur.

Les données à archiver sont ajoutées au fichier au fur et à mesure de son exécution, et plus généralement, au fur et à mesure des exécutions du programme.

Voici la structure du fichier d'historique:

```
Date/Heure Numéro_de_Feux Événement
```

Date/Heure: jj/mm/aaaa hh:mm:ss

Numéro_de_Feux: 1, 2, 3 ou 4 selon le feu.

Événement: « Appui BP » ou « Detecteur »

Les espaces entre chaque champ est à remplacer par le caractère séparateur choisi. Ce caractère est paramétrable dans le programme.

Le contenu du champ « Événement » peut facilement être modifié dans le programme. Une autre possibilité aurait été de faire un second ficher de configuration, ou alors écrire à la suite de celui nécessaire à la régulation des temps des feux.

Le programme, pour effectuer son travail, se connectera soit à la maquette du carrefour, soit à un simulateur fourni. Une adresse IP sera utilisée pour se connecter au carrefour ou à la maquette. Pour utiliser la maquette, il faudra lancer le programme depuis une console et taper le caractère « M » juste après son nom. Les adresses étant fixes, pas besoin de les lui donner explicitement.

2. Environnement

Le développement d'un tel programme nécessite l'utilisation de plusieurs fonctions permettant de « discuter » sur le réseau avec la maquette ou le simulateur, et donc d'en connaître précisément la conception. Heureusement, une bibliothèque C#, CmdCrf.dll nous est donnée et contient tout le nécessaire pour réaliser le programme, sans devoir développer la communication sur le réseau.

Pour éviter que le carrefour ne se mette dans un état « travaux » (tous les feux oranges allumés clignotants), il est nécessaire de communiquer régulièrement avec le carrefour (maquette ou simulateur). Un minimum de 10 ms pour surveiller les événements et communiquer avec le carrefour semble acceptable.

La gestion des événements du carrefour, à savoir un appui sur les bouton-poussoirs piéton et la détection des véhicules s'opérera en lisant les données sur le réseau puis en allant chercher les booléens « BP » et « Detecteur » de chaque feux, sachant qu'une fois ces données récupérées, elles sont effacées du carrefour. Il conviendra de traiter ces informations avant de communiquer avec le carrefour.

Il existe aussi un tableau répertoriant les quatre feux du carrefour, ce qui sera très pratique pour un traitement des données ou une boucle pourra être utilisée.

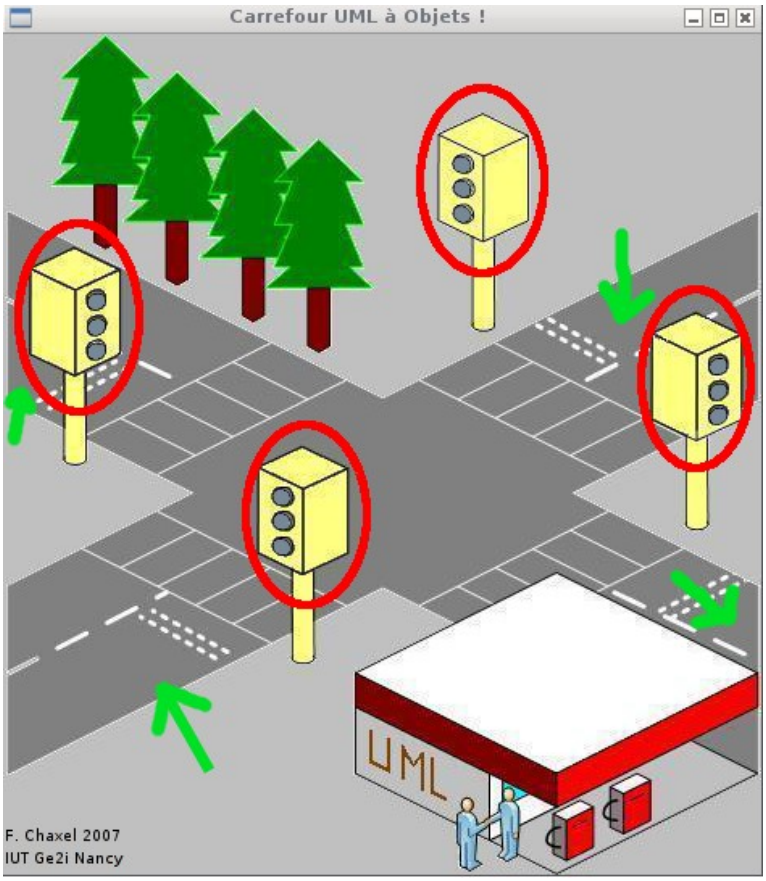
3. Maquette cible

Nous l'avons dit précédemment, il existe deux modes d'utilisation du programme. L'un repose sur l'utilisation d'un simulateur graphique fourni, et l'autre sur l'utilisation d'une maquette embarquant un micro-contrôleur avec une connexion Ethernet.

Pour utiliser le simulateur, il suffira de le lancer, et il utilisera l'adresse IP locale par excellence : 172.0.0.1

Sinon, une fois la maquette connectée au routeur de votre réseau local, vous pourrez vous-y connecter avec l'adresse suivante : 172.20.54.111

Pour l'utilisation du simulateur, il vous suffira de cliquer comme le montre l'image pour activer les BP ou les Détecteurs :



4. Fonctionnement globale

On commence par vérifier si les fichiers de configuration et d'historique peuvent être utilisés :

- Si le fichier d'historique ne peut pas être créé, alors le programme va s'arrêter sans rien faire du tout.
- S'il ne trouve pas le fichier, alors le programme écrit dans le fichier d'historique : « Erreur Fichier Config.txt introuvable » puis s'arrête.

Si tout va bien, le programme se contentera de lire le fichier de configuration et d'en extraire les informations intéressantes, en l'occurrence, les nombres.

Une fois ces informations récupérées, on se contentera de regarder régulièrement les événements du carrefour pour savoir s'il faut passer la temporisation à Tmini ou laisser à Tmax, s'il faut ensuite changer de voie et ainsi de suite.

Dans le cas où la communication entre la maquette (ou le simulateur) est perdue en cours de route, écrit « Erreur Connexion » dans l'historique puis se ferme.

À chaque fois qu'un véhicule est détecté ou qu'un piéton appuie sur un bouton, la date suivie du numéro de feux et l'événement associé sera enregistré.

Le programme peut fonctionner à l'infinie (par défaut) comme pour un nombre de cycles donné (passage de paramètres au lancement du programme). À cause des limitations de taille des variables, nous ne pourrons faire au maximum qu'un peu plus de 2 milliards de cycles. Mais à ce niveau là, autant aller à l'infinie et au-delà.

Un cycle correspond à un changement de voie. C'est à dire que si on a le vert sur la voie 1, une fois que le vert sera sur la voie 2 (et donc au rouge sur la voie 1) cela fait un cycle.

5. Fonctionnement détaillé

5.1. Lancement du programme

Pour comprendre ce qui suit, je vous conseil d'ouvrir le fichier Main.cs ainsi que Carrefour.cs.

Observons pour le moment Main.cs :

Les premières 5 premières lignes de code déclarent quelques variables nécessaires à l'initialisation du carrefour. On y trouve :

- Un objet « Crossroads »
- Un string « path »
- Un string « ip »
- Un entier 32bit « doCycles »
- Un char « sepChar »

Le premier correspond à la classe se trouvant dans le fichier Carrefour.cs. C'est de cet objet que nous allons nous servir pour commander le carrefour.

Le premier « string » indique où trouver les fichiers de configuration et d'historique. En fait, c'est un chemin vers le répertoire devant contenir ces fichiers. L'adresse finale est complétée plus loin.

Le second sélectionne une adresse IP par défaut, dans le cas où aucun paramètre n'est passé au lancement du programme.

« doCycles » est une valeur par défaut indiquant que le programme continuera à l'infinie car ici initialisée à 0.

« sepChar » recueille le caractère séparateur dans le fichier historique. Bien entendu, il ne devra pas être un espace... aucune protection contre ça, mais il faut vraiment le vouloir pour le faire.

5.1.1. Paramètres programme

La suite du main est un traitement extrêmement basique des paramètres programme. C'est à dire que, depuis la console Windows (ou Linux/BSD/Mac OS, grâce à Mono) on va lancer le programme comme suit :

```
nom_du_programme.exe param0 param1 param2
```

Liste des arguments :

- args[0] : maquette (M) et simulateur (autre)
- args[1] : nombre de cycles. 0 = nombre infinit.
- args[2] : caractère séparateur. Écrire « tab » pour obtenir une tabulation, sinon pour certains caractères pouvant être interprétés par le shell, mettre entre guillements.
- args[3] : option facultative sous Windows, obligatoire sous Unix.

Le test de chacun des arguments dépend de la présence du précédent. Sinon nous aurons des valeurs par défaut.

Ensuite il suffit de lancer la fonction RunCrossroads(doCycles) et le carrefour démarre.

5.2. L'objet « Crossroads »

Les variables précédemment déclarées vont maintenant nous servir à initialiser l'objet « Crossroads » (Carrefour en anglais).

Cet objet contient toutes les fonctions et variables pour gérer le carrefour. Il peut être compilé à part pour créer une bibliothèque dans le même genre de CmdCrf.dll !

Lors de la création de l'objet, un certain nombre de variables sont initialisées. Allez voir le constructeur de Crossroads dans Carrefour.cs.

D'abord on traite les fichiers, seulement ensuite on traite les variables. Inutile de le faire si on ne peut pas avoir accès aux fichiers.

Les temps de passages des voies sont stockées dans le tableau « this.times ». Le mot clé « this » est ajouté partout où les variables et fonctions sont internes à la classe.

« this.curWay » indique la voie actuellement au vert. Deux autres valeurs, « this.way1 » et « this.way2 » permettent de faire des tests sur cette variable. Pas vraiment nécessaire puisque c'est un booléen, mais plus clair.

« this.memEventFlag » est un booléen qui sert juste à dire ceci : « il s'est déjà produit un événement sur le carrefour, j'attends qu'on passe à la voie suivante pour me taire. ». Pour le faire taire, il faut le mettre à false.

La fonction des autres variables sont facilement compréhensibles.

5.3. Algorithme de fonctionnement du carrefour

Le programme se déroule dans l'ordre suivant :

- Appel de RunCrossroads(int doCycles)
  - Initialisation de variables internes à la fonction :
    - Nombre de cycles = 0 ;
    - Nombre de millisecondes écoulées = 0 ;
    - Initialisation de infinite à false ;
    - Positionnement des couleurs deux feux :
      - Voie courant au vert ;
      - Autre voie au rouge ;
    - Lecture de la configuration des temps ;
    - Écriture des données sur le réseau ;
  - Si doCycles == 0 :
    - Alors infinite = true → boucle infinie ;
    - Boucle tant que Nombre de cycles < doCycles OU infinite == true
      - Si on a atteint le temps maximum configuré (voir constructeur) :
        - Remettre à zéro le drapeau des événements ;
        - TmaxHit() → configure le carrefour dans le cycle suivant ;
      - Sinon si pas d'événements déjà enregistré et tMax pas atteint :
        - WayEvents() → Regarde s'il y a eu des événements et met le drapeau des événements à true s'il s'en est produit, écriture dans l'historique.

Les fonctions sont suffisamment bien commentées dans le code pour pouvoir comprendre ce qu'il s'y passe. À chaque fois vous trouverez une liste des paramètres d'entrée (s'il y en a) et, après un saut de ligne, la valeur de retour (s'il y en a).

Comprenez cependant que la variable this.tMax prend plusieurs valeurs au cours de l'exécution du programme. Ces valeurs sont changées lors que les conditions de passages sont remplies.

Les variables déclarées en tête de la classe Crossroads ne sont PAS des variables globales mais des membres. Même en C on est pas aussi suicidaire que ça, les structures nous facilitent les choses.

6. Conclusion

Ayant déjà fait de la programmation en Script Bash/SH, C, C++, PHP, Python et Java, ce projet m'a surtout permit de connaître un autre langage, et aussi de partager mes connaissances avec les personnes du groupe dans lequel je suis. Le plus difficile cependant a été la gestion du temps. Enfin il m'a permit aussi de m'investir dans la lisibilité du code et une documentation à mettre à jour sans cesse dans le code afin que quiconque sachant programmer un peu puisse les comprendre. Quand on est le seul à lire son code, on en a que faire du reste du monde.

Le code est publié sous licence BSD car pour des raisons pratiques, j'héberge tous mes projets de programmation sur internet. S'il advenait que mon code se retrouve dans les mains d'une autre personne et utilise mon code pour son projet, je peux prouver que j'en suis l'authentique auteur d'origine. Le choix de la licence c'est une autre histoire.

Je suis à disposition par mail pour d'éventuels commentaires et questions à cette adresse :

florent.peterschmitt.1@etumail.uhp-nancy.fr