Ingénierie des modèles de simulation

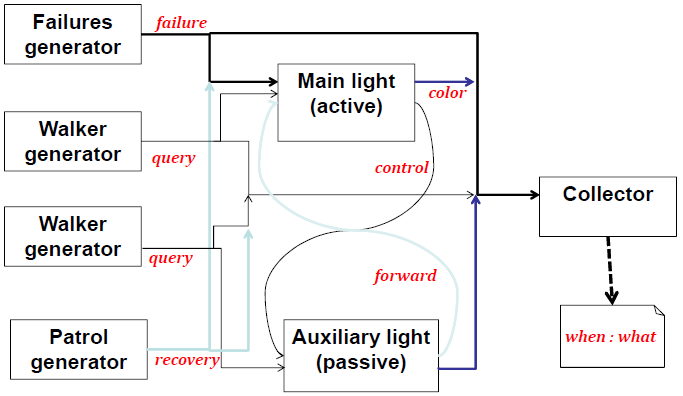
# Objectifs

L’objectif du TP est de simuler le comportement des feux de circulation à une intersection. Il y a quatre feux tricolores, deux principaux et deux secondaires. Les feux secondaires dépendent entièrement des feux principaux pour leur fonctionnement (les feux principaux indiquent aux secondaires quand changé de couleurs).

Les feux principaux peuvent tomber en panne. Dans ce cas les feux secondaires sont en panne aussi. La réparation se fait lorsqu’une patrouille passe, sachant que la patrouille passe régulièrement.

Enfin, les deux types de feux doivent gérer les demandes de piétons voulant traverser. Au-delà d’un certain nombre de demandes, les feux changent de couleur. Dans le cas des feux secondaires, ceux-ci ne changent pas directement de couleur mais ils envoient une requête aux feux principaux qui changeront alors de couleur.

# Le modèle



Le modèle du simulateur est représenté dans la figure ci-dessus. Il est composé de sept sous modèles. Il y a un sous modèle pour générer les pannes des feux principaux, un sous modèle pour simuler les passages de patrouille et deux sous modèles pour simuler les demandes des piéton, un pour les feux principaux et un pour les feux secondaires. Ensuite, il y a un collecteur qui est utilisé pour écrire les évènements du système dans un fichier. Les deux derniers sous systèmes sont celui des feux principaux et celui des feux secondaires.

Le sous modèle des feux principaux prend en entrée les sorties des générateurs de panne et de patrouille et d’un des générateur de demande de piéton. Il prend aussi en entrée les requêtes de changement de couleurs des feux secondaires (lorsque les piétons veulent traverser la voie secondaire). Le sous modèle des feux secondaires prend entrée la sortie du deuxième générateur de demande de piéton et une sortie du sous modèle des feux principaux, qui indique à quels moments changer de couleur. Les deux sous modèles de feux comportent deux sorties, l’une est dirigée vers l’autre modèle pour l’échange de message comme expliqué avant, l’autre est dirigée vers le collecteur pour inscrire tous les changements d’états dans un fichier.

# Les sous modèles

## Les pannes

Mfail = <X, Y, S, δint, δext, λ, ta>

X = ∅

Y = {0, 1}

S = {STEADY}

* δint : S → S

STEADY → STEADY

* δext : non définie
* λ : S → Y

STEADY → Udisc(0, 1)

* ta : S → ℝ+

STEADY → Ucont(4, 20)

Le générateur de pannes ne contient qu’un état interne dont la durée est comprise entre 4 et 20. Lors du changement d’état, un message est envoyé, soit il y a une panne (« Failure ») soit il n’y en a pas (en pratique, si il n’y a pas de panne, il n’y a pas d’envoi de message).

## Les patrouilles

Mpat = <X, Y, S, δint, δext, λ, ta>

X = ∅

Y = {1}

S = {STEADY}

* δint : S → S

STEADY → STEADY

* δext : non définie
* λ : S → Y

STEADY → 1

* ta : S → ℝ+

STEADY → 15

Le générateur de patrouilles ne contient qu’un état interne dont la durée est de 15 unités de temps. Lors du changement d’état, un message est envoyé pour indiquer qu’une patrouille passe (« Recovery »).

## Les piétons

Mwalk = <X, Y, S, δint, δext, λ, ta>

X = ∅

Y = {0, 1}

S = {STEADY}

* δint : S → S

STEADY → STEADY

* δext : non définie
* λ : S → Y

STEADY → Udisc(0, 1)

* ta : S → ℝ+

STEADY → Ucont(0,4)

Le générateur de pannes ne contient qu’un état interne dont la durée est comprise entre 0 et 4. Lors du changement d’état, un message est envoyé, soit il y a un piéton (« Walker ») soit il n’y en a pas (il n’y a alors pas d’envoi de message).

## Le collecteur

Mcoll = <X, Y, S, δint, δext, λ, ta>

X = {G, Y, R, O, W}

Y = ∅

S = {FILE}

* δint : non définie
* δext : Q × X → S

((FILE, e), x) → FILE / FILE = FILE + {x}

* λ : non définie
* ta : S → ℝ+

STEADY → +∞

Le collecteur reçoit les messages de sortie des modèles des feux et les écrits dans un fichier. Il n’a qu’un seul état d’une durée infini, donc il n’y a pas de changements d’état internes mais uniquement externes (à la réception d’un message).

## Les feux principaux

Mmain = <X, Y, S, δint, δext, λ, ta>

X = {Failure, Recovery, Walker, Auxiliary}

Y = {Green, Yellow, Red, Off, Walker}

S = {Green, Yellow, Red1, Red2, Off1, Off2, Walker} × ℝ+

* δint : S → S

(Green, σ) → (Yellow, 1)

(Yellow, σ) → (Red1, 2)

(Red1, σ) → (Red2, 1)

(Red2, σ) → (Green, 5)

(Off1, σ) → (Off2, +∞)

(Walker, σ) → (Yellow, 1)

(Walker, σ) → (Red1, 2)

(Walker, σ) → (Red2, 1)

(Walker, σ) → (Green, 5)

* δext : Q × X → S

(((s, σ ), e), Failure) → (Off1, 0)

(((Off2, σ ), e), Recovery) → (Green, 5)

(((s, σ ), e), Walker) → (Walker, 0)

(((s, σ ), e), Auxiliary) → (Red1, 2)

(((s, σ ), e), Auxiliary) → (Red2, 1)

* λ : S → Y × Y

(Green, σ) → (Green, Red)

(Yellow, σ) → (Yellow, ∅)

(Red1, σ) → (Red, Green)

(Red2, σ) → (∅, Yellow)

(Off1, σ) → (Off, Off)

(Walker , σ) → (Walker, ∅)

* ta : S → ℝ+

(s, σ) → σ

La durée du feu vert est de 5 unités de temps, celle du feu orange est de 1 et celle du feu rouge est de 3, décomposée en deux étapes, une première qui dure 2 et une deuxième qui dure 1. La deuxième étape du feu rouge permet d’envoyer un message aux feux secondaires pour les faire passer à l’orange.

Si le simulateur reçoit le message « Failure » et qu’il n’est pas déjà en panne, il passe à l’état « Off1 » pour envoyer un message au collecteur puis il passe immédiatement à l’état « Off2 » pour une durée infinie. S’il reçoit « Recovery » et qu’il est en panne, il passe au vert et se remet à fonctionner normalement. Lorsqu’un piéton demande à traverser, l’état dans lequel se trouve le simulateur est mémoriser (ainsi que la durée écoulée depuis le dernier changement d’état), le simulateur passe à l’état « Walker » pour transmettre un message au collecteur puis le simulateur repasse dans l’état sauvegardé pour la durée restante de cet état. S’il y a beaucoup de demande, au lieu de retourner à l’état sauvegardé, le simulateur passe directement à l’état « Yellow ». Enfin, le message « Auxiliary » indique que les feux secondaires ont beaucoup de demande de piéton, dans ce cas le feu passe soit à l’état « Red2 » s’il était en « Red1 », soit à l’état « Green » s’il était à l’état « Red2 », ce qui permet de réduire le durée soit des feux secondaires verts, soit des feux secondaires oranges (en fonction de l’état dans lequel ils se trouvent).

L’état des feux secondaires est géré par les feux principaux, ceux-ci ont donc deux sorties comme on peut le voir dans la fonction lambda. La première sortie est utilisée par le collecteur, la deuxième par les feux secondaires.

## Les feux secondaires

Maux = <X, Y, S, δint, δext, λ, ta>

X = {Failure, Walker, Green, Yellow, Red}

Y = {Green, Yellow, Red, Off, Walker, Auxiliary}

S = {Green, Yellow, Red, Off, Walker} × ℝ+

* δint : S → S

(Walker, σ) → (Yellow, +∞)

(Walker, σ) → (Red, +∞)

(Walker, σ) → (Green, +∞)

* δext : Q × X → S

(((s, σ ), e), Failure) → (Off, +∞)

(((s, σ ), e), Walker) → (Walker, 0)

(((s, σ), e), Yellow) → (Yellow, +∞)

(((s, σ ), e), Red) → (Red, +∞)

(((s, σ ), e), Green) → (Green, +∞)

* λ : S → Y × Y

(Green, σ) → (Green, ∅

(Yellow, σ) → (Yellow, ∅)

(Red, σ) → (Red, ∅)

(Off, σ) → (Off, ∅)

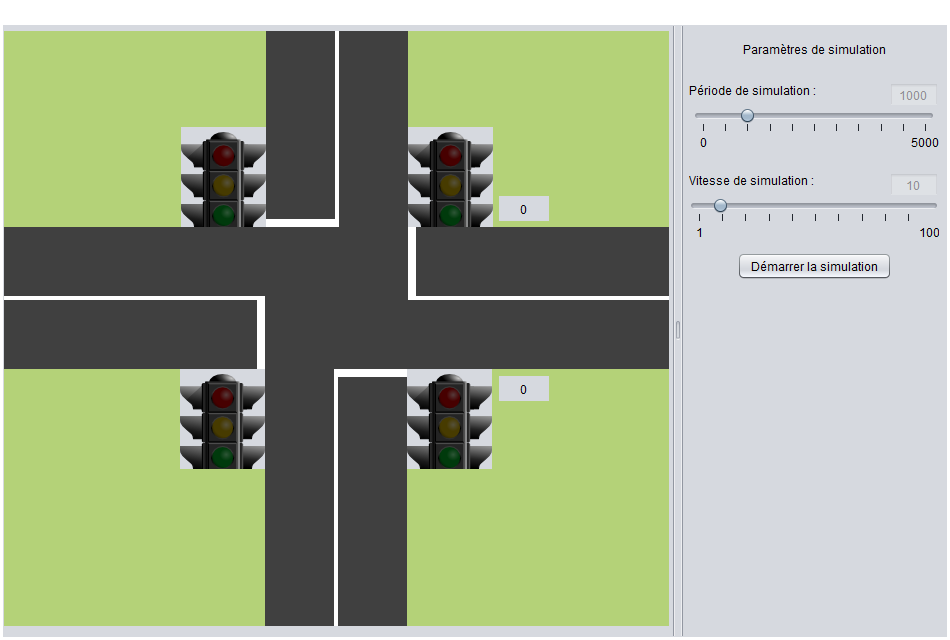
(Walker , σ) → (Walker, Auxiliary)

* ta : S → ℝ+

(s, σ) → σ

L’état des feux secondaires est contrôler par les feux principaux, donc les états ont des durées infinies. La seule exception est l’état « Walker » dans lequel le simulateur est après avoir reçu un message du simulateur de piétons. Le fonctionnement de cet état est le même que dans le simulateur des feux principaux. Comme pour les feux principaux, il y a deux sorties, la première pour le collecteur, la deuxième pour les feux principaux.

# L’interface graphique



L’interface graphique est composée de deux parties. A gauche, il y a une représentation du croisement étudié. La route principale est la route nord/sud. Le nombre de requête de piétons pour traverser la route principale est affiché à droite du feu sud, même chose pour la route secondaire à droite du feu est. La partie de droite permet de choisir la durée de la simulation, entre 0 et 5000 unités de temps, ainsi que la vitesse de l’affichage (entre x1 et x100).