TP #1 – SYSTÈME ROUTIER

8INF259 -- Structure de données

Thématique

Dans ce projet, vous allez concevoir un système simulant la gestion du trafic routier à des intersections. Votre mission est de gérer le flux de véhicules sur différentes routes et à travers diverses intersections avec des règles de priorité distinctes. L'objectif est de minimiser le temps d'attente total des véhicules tout en respectant les règles de circulation et en évitant les blocages.

Objectifs pédagogiques

Ce travail pratique permet de :

1. **Comprendre les structures de données dynamiques**, comme les files pour gérer le flux de véhicules sur les routes.
2. **Gérer la logique algorithmique**, comme la coordination des feux de circulation et la gestion des priorités aux intersections.
3. **Simuler un système fonctionnel** où vous implémentez la logique de circulation, de priorisation et de gestion du temps d'attente.
4. **Analyser la complexité algorithmique** en comparant différentes stratégies de gestion du trafic.

En réalisant ce TP, vous développerez des compétences en **algorithmique**, **gestion de flux** et **gestion de la mémoire dynamique**.

Exigences

Introduction

Vous devez implémenter plusieurs structures pour simuler un système de gestion du trafic routier. Ce système gère des véhicules circulant sur différentes routes et traversant des intersections selon des règles de priorité variables. Vous développerez les structures nécessaires pour gérer les véhicules, les routes, et les intersections.

Un exemple de main.cpp est fourni plus bas, mais vous pouvez le modifier à votre guise, du moment que les consignes sont respectées.

Structures et classes à développer

1. **Structure Vehicle (ou classe)**
    - **Rôle :** Représenter un véhicule et ses attributs.
    - **Attributs :**
        - std::string id : Identifiant unique du véhicule (ex: "V001").
        - std::string type : Type de véhicule ("Voiture", "Camion", "Moto").
        - int waitTime : Temps d'attente actuel (en tours).
        - std::string destination : Direction de destination ("Nord", "Sud", "Est", "Ouest").
    - **Méthodes suggérées :**
        - Constructeur avec initialisation de tous les attributs.
        - Méthode pour incrémenter le temps d'attente (increaseWaitTime).
        - Méthode pour afficher les informations du véhicule (display).

2. **Enum TrafficLightState**
    - **Rôle :** Définir les états possibles d'un feu de circulation.
    - **Détails :**
        - Énumération des états :
            - RED
            - GREEN
            - YELLOW (optionnel)

3. **Enum IntersectionType**
    - **Rôle :** Définir les types d'intersections.
    - **Détails :**
        - Énumération des types :
            - PRIORITY_LIGHT : Feu avec gestion dynamique de priorité
            - FIXED_LIGHT : Feu avec cycle fixe
            - FOUR_WAY_STOP : Intersection avec 4 stops

4. **Structure/Classe pour gérer une file de véhicules**
    - **Rôle :** Implémenter le TAD file pour gérer les véhicules sur une route.
    - **Fonctionnalités requises :**
        - Ajout d'un véhicule (enqueue).
        - Retrait du premier véhicule (dequeue).
        - Consultation du premier véhicule sans le retirer (peek).
        - Vérification si la file est vide.
        - Accès au nombre de véhicules dans la file.
        - Possibilité de parcourir tous les véhicules (pour affichage et mise à jour du temps d'attente).

5. **Structure Road (ou classe)**
    - **Rôle :** Représenter une route avec sa file de véhicules.
    - **Attributs :**
        - std::string name : Nom de la route (ex: "Route Nord").
        - FILEDEVEHICULE vehicles : File de véhicules sur cette route.
        - std::string direction : Direction de la route ("Nord", "Sud", "Est", "Ouest").
    - **Méthodes suggérées :**
        - addVehicle(Vehicle v) : Ajoute un véhicule à la file.
        - Vehicle* getNextVehicle() : Retourne et retire le premier véhicule.
        - void increaseAllWaitTimes() : Incrémente le temps d'attente de tous les véhicules.
        - void display() : Affiche l'état de la route.
        - int getVehicleCount() : Retourne le nombre de véhicules sur la route.

6. **Structure Intersection (ou classe)**
    - **Rôle :** Gérer une intersection avec ses routes connectées et ses règles de circulation.
    - **Attributs :**
        - std::string name : Nom de l'intersection.
        - IntersectionType type : Type d'intersection.
        - Road* northRoad : Route venant du nord.
        - Road* southRoad : Route venant du sud.
        - Road* eastRoad : Route venant de l'est.
        - Road* westRoad : Route venant de l'ouest.
        - TrafficLightState northSouthLight : État du feu Nord-Sud (si applicable).
        - TrafficLightState eastWestLight : État du feu Est-Ouest (si applicable).
        - int cycleCounter : Compteur pour gérer les cycles de feux.
        - int greenDuration : Durée minimale du vert (en tours).

        - int maxGreenDuration : Durée maximale du vert (en tours, pour PRIORITY_LIGHT).
    - **Méthodes suggérées :**
        - processTurn() : Traite un tour de circulation selon le type d'intersection.
        - updateLights() : Met à jour l'état des feux.
        - void display() : Affiche l'état de l'intersection.

7. **Structure/Classe TrafficSystem**
    - **Rôle :** Coordonner l'ensemble du système de circulation.
    - **Attributs :**
        - std::vector<Intersection*> intersections : Liste des intersections.
        - int totalWaitTime : Temps d'attente cumulé de tous les véhicules.
        - int processedVehicles : Nombre de véhicules ayant traversé.
    - **Méthodes suggérées :**
        - addIntersection(Intersection* intersection) : Ajoute une intersection au système.
        - void processTurn() : Effectue un tour pour toutes les intersections.
        - void displayState() : Affiche l'état de toutes les intersections.
        - int getTotalWaitTime() : Retourne le temps d'attente total cumulé.
        - int getProcessedVehicles() : Retourne le nombre de véhicules traités.

Attentes fonctionnelles

Gestion des files de véhicules :

- Les véhicules doivent être gérés dans des files (FIFO) sur chaque route.
- Le temps d'attente de chaque véhicule doit être incrémenté à chaque tour où il n'avance pas.

Types d'intersections :

- **PRIORITY_LIGHT (Feu avec priorité dynamique) :**
    - Les feux changent en fonction de la densité de trafic.
    - La direction avec le plus de véhicules en attente obtient le feu vert.
    - Durée minimum de vert : configurable (ex: 2 tours).
    - Durée maximum de vert : configurable (ex: 5 tours) pour éviter la monopolisation.
    - Quand Nord-Sud est vert, Sud-Nord est aussi vert (circulation bidirectionnelle).
- **FIXED_LIGHT (Feu fixe) :**
    - Les feux alternent selon un cycle fixe (ex: 3 tours vert Nord-Sud, puis 3 tours vert Est-Ouest).
    - Quand Nord-Sud est vert, Sud-Nord est aussi vert (circulation bidirectionnelle).
- **FOUR_WAY_STOP (4 stops) :**
    - Un seul véhicule passe à la fois, en rotation entre les 4 directions.
    - Priorité à celui qui attend depuis le plus longtemps (premier arrivé, premier servi).

Gestion du temps d'attente :

- Chaque véhicule qui ne bouge pas voit son temps d'attente augmenter.
- Le système doit minimiser le temps d'attente total.

Analyse de performance (OBLIGATOIRE)

IMPORTANT : Vous devez créer un fichier RESULTATS.md à la racine de votre projet contenant vos résultats d'analyse.

Vous devez tester votre système avec **120 véhicules** répartis selon trois configurations différentes :

Configurations à tester :

1. **EVEN (Trafic équilibré)** : 30 véhicules dans chaque direction (N=30, S=30, E=30, O=30)
2. **ONEWAY (Un axe vide)** : 60 véhicules Nord, 60 véhicules Sud, 0 véhicule Est, 0 véhicule Ouest (N=60, S=60, E=0, O=0)
3. **UNBALANCED (Trafic déséquilibré)** : 50 véhicules Nord, 50 véhicules Sud, 10 véhicules Est, 10 véhicules Ouest (N=50, S=50, E=10, O=10)

Types d'intersections à tester :

Pour chaque configuration, testez les trois types d'intersections :

- PRIORITY_LIGHT (min: 2 tours, max: 5 tours)
- FIXED_LIGHT (cycle: 3 tours)
- FOUR_WAY_STOP

Métrique à mesurer :

- **Temps d'attente moyen** = Temps d'attente total / Nombre de véhicules traités

Format du fichier RESULTATS.md :

Votre fichier doit contenir un tableau markdown avec vos résultats et une analyse :

```
# Résultats d'analyse - TP2 Gestion du trafic

## Tableau des résultats

| Configuration | PRIORITY_LIGHT | FIXED_LIGHT | FOUR_WAY_STOP |

|---------------|----------------|-------------|---------------|

| EVEN          | X.XX           | X.XX        | X.XX          |

| ONEWAY        | X.XX           | X.XX        | X.XX          |

| UNBALANCED    | X.XX           | X.XX        | X.XX          |

*Temps d'attente moyen en tours par véhicule (120 véhicules total)*

## Analyse

### Meilleur algorithme par configuration

- **EVEN** : [Indiquer le meilleur type et expliquer pourquoi]

- **ONEWAY** : [Indiquer le meilleur type et expliquer pourquoi]

- **UNBALANCED** : [Indiquer le meilleur type et expliquer pourquoi]
```

```
### Observations générales

[Vos observations sur les forces et faiblesses de chaque type d'intersection]

### Complexité algorithmique

[Analyse de la complexité en temps et en espace de votre implémentation]

### Conclusion

[Synthèse de vos résultats : quel type d'intersection est le plus efficace et dans quelles situations]
```

Note : Remplacez les X.XX par vos valeurs mesurées. Soyez précis dans vos analyses et justifications.

Format de rendu

Pour être valide, il faut rendre un unique fichier .zip **comportant le dossier racine**. Ce fichier devra être nommé : « TP1_NOM_PRENOM.zip » en remplaçant bien entendu les « NOM » et « PRENOM » par vos noms et prénoms respectifs. Les fichiers devront directement être à la racine du fichier .zip (si vous avez des sous-dossiers, n'inclure que le code source, et garder votre structure).

Le fichier RESULTATS.md doit être présent à la racine du projet.

Date de rendu

Le projet devra être remis avec **un seul fichier .zip au plus tard le 13 février 2026 à 23h59**. Le rendu se fera sur Moodle via l'option créée à cet effet.

Conseils

- Commencez par implémenter la file de véhicules avant tout.
- Testez chaque type d'intersection séparément.
- Pour les tests de performance, désactivez les affichages détaillés (commentez les std::cout dans les méthodes processTurn).
- Notez précisément vos résultats pour chaque configuration testée.
- Réfléchissez aux raisons des différences de performance entre les algorithmes.

Grille de notation

Structure Vehicle (5%)

- Implémentation correcte des **attributs** : id, type, waitTime, destination.
- Méthodes essentielles fonctionnelles.

Enums TrafficLightState et IntersectionType (5%)

- Définition correcte des états de feux et types d'intersections.

Système de file personnalisé (15%)

- Création d'une structure/classe dynamique pour gérer les files (pas d'utilisation directe de std::queue).
- Fonctionnalités requises complètes et fonctionnelles.
- Gestion correcte de la mémoire.

Structure Road : Gestion des véhicules (10%)

- Organisation correcte des véhicules dans la file.
- Toutes les méthodes fonctionnelles.

Structure Intersection : Feux fixes (FIXED_LIGHT) (10%)

- Implémentation correcte du cycle fixe.
- Gestion bidirectionnelle (Nord-Sud ensemble, Est-Ouest ensemble).
- Alternance correcte des feux selon la durée définie.

Structure Intersection : Feux avec priorité (PRIORITY_LIGHT) (15%)

- Calcul de la priorité basé sur le nombre de véhicules.
- Respect de la durée minimum avant changement.
- Respect de la durée maximum pour éviter la monopolisation.
- Gestion bidirectionnelle correcte.

Structure Intersection : 4 stops (FOUR_WAY_STOP) (10%)

- Rotation correcte entre les 4 directions.
- Respect de l'ordre d'arrivée (temps d'attente).
- Un seul véhicule passe à la fois.

Structure TrafficSystem : Coordination (10%)

- Gestion correcte de multiples intersections.
- Calcul du temps d'attente total cumulé.
- Comptabilisation des véhicules traités.

Analyse de performance et fichier RESULTATS.md (15%)

- Présence du fichier RESULTATS.md à la racine.
- Tableau complet avec les 9 résultats (3 configurations × 3 types).
- Analyse pertinente et justifiée des résultats.
- Réflexion sur la complexité algorithmique.
- Conclusion argumentée sur l'efficacité des différents types.

Robustesse (5%)

- Gestion correcte des **cas limites**.
- Pas de fuites mémoire.
- Code qui compile et s'exécute sans erreur.

P.S. Chaque structure peut être remplacée par une classe à la volonté de l'étudiant.

Annexe -- main.cpp

Vous êtes complètement libres sur votre main, mais il faut être capable de changer les paramètres facilement (commenter/décommenter, assigner une valeur à une variable/constante en haut du main, etc.).

Voici un exemple du contenu de mon main (à titre informatif) :

```
int main() {
    #ifdef _WIN32
    // Fix UTF-8 encoding on Windows
    SetConsoleOutputCP(CP_UTF8);
    SetConsoleCP(CP_UTF8);
    #endif

    TrafficSystem system;

    // Create roads
    Road northRoad{"Route Nord", "Nord"};
    Road southRoad{"Route Sud", "Sud"};
    Road eastRoad{"Route Est", "Est"};
    Road westRoad{"Route Ouest", "Ouest"};

    // Choose one of the following scenarios:

    // Scenario 1: Equal traffic on all roads (30 vehicles each = 120 total)
    addEqualVehicles(northRoad, southRoad, eastRoad, westRoad, 30);

    // Scenario 2: Empty East-West axis (60 vehicles each on N-S = 120 total)
    // addEmptyAxis(northRoad, southRoad, eastRoad, westRoad, 60);

    // Scenario 3: Unbalanced traffic (50 vehicles N-S, 10 vehicles E-W = 120
total)
    // addUnbalancedVehicles(northRoad, southRoad, eastRoad, westRoad, 50, 10);

    // Create intersection - choose one type:

    // Type 1: Priority Light (dynamic based on traffic)
    Intersection intersection1{
        "Intersection Principale",
        PRIORITY_LIGHT,
        &northRoad,
        &southRoad,
        &eastRoad,
        &westRoad,
        2,  // minimum green duration
        5   // maximum green duration
    };

    // Type 2: Fixed Light (alternating cycle)
    // Intersection intersection1{
    //     "Intersection Principale",
    //     FIXED_LIGHT,
    //     &northRoad,
    //     &southRoad,
    //     &eastRoad,
    //     &westRoad,
    //     3  // green duration for each direction
    // };

    // Type 3: Four-Way Stop (one vehicle at a time)
```

```
    // Intersection intersection1{
    //     "Intersection Principale",
    //     FOUR_WAY_STOP,
    //     &northRoad,
    //     &southRoad,
    //     &eastRoad,
    //     &westRoad
    // };

    system.addIntersection(&intersection1);

    // Simulation
    for (int turn = 1; system.hasVehicles(); turn++) {
        // Uncomment the following lines to see detailed simulation
        // std::cout << "========== Tour " << turn << " ==========" <<
std::endl;
        // system.displayState();
        system.processTurn();
        // std::cout << std::endl;
    }

    std::cout << "========== Résultats finaux ==========" << std::endl;
    std::cout << "Véhicules traités: " << system.getProcessedVehicles() <<
std::endl;
    std::cout << "Temps d'attente total: " << system.getTotalWaitTime() << "
tours" << std::endl;
    std::cout << "Temps d'attente moyen: "
              << (system.getProcessedVehicles() > 0 ?
                  (float)system.getTotalWaitTime() /
system.getProcessedVehicles() : 0)
              << " tours/véhicule" << std::endl;

    return 0;
}
```

Dans cet exemple, je modifiais les lignes de scénario et l'intersection concernée pour mes différentes exécutions afin de compiler le Markdown final.

Comme mentionné plus tôt, assurez-vous de tester toutes les configurations demandées et de remplir le fichier RESULTATS.md avec vos analyses !