MOUANOU GUICHEL
BENMOULOUD MEHDI
MOHAMEDATNI AYA

RAPPORT PROJET INFORMATIQUE

THEME:

MODÉLISATION D'UN TRAFIC ROUTIER

PLAN:

I/INTRODUCTION

II/ ANALYSE DU PROBLÈME

1. SEMAINE 1.2.3: ETUDE DU SUJET + DÉFINITION DES AGENTS

   1.1. Étudier le sujet

   1.2. Comprendre le fonctionnement d'un agent

2. SEMAINE 4/5: ETUDE DES INTERFACES GRAPHIQUES

   1.1. Construction de notre première fenêtre avec Jframe

   1.2. Première animation sur notre fenêtre

III/ MODELISATION

1.SEMAINE 6: CLASSE + HÉRITAGE

   1.1. Classes choisies et lien entre elles

   1.2. Code de ces classes

2. SEMAINE 7/8: ENVIRONNEMENT AVEC JPANEL + GESTION DES IMAGES

   1.1. Création d'un panneau pour gérer les placements

   1.2. Etude de l'insertion des images et construction en 2D

IV/ APPROFONDISSEMENT

1.SEMAINE 7/8/9 : ETUDE DES COMPORTEMENTS ENTRE LES AGENTS

   a. Comportement des agents qui se déplacent à l'horizontal

   b. Comportement des agents qui se déplacent à la verticale

2. SEMAINE 9/10: MIS EN PLACE DE TOUS LES AGENTS + RAPPORT

   a. Réglage des problèmes engendré par la mise en place de beaucoup d'agents

   b. Finir le rapport

V/ BILAN

I/ INTRODUCTION

Un système multi-agent est composé d'entités informatiques dont chacun a une activité et des informations propres, appelées **agents**, qui évoluent et interagissent dans **un environnement commun**.

La notion d'interaction entre agents est essentielle car chacun d'eux est impliqué dans une **dynamique commune**, au lieu d'évoluer indépendamment des autres.
Cette répartition implique que chaque agent puisse effectuer localement la tâche qui lui est assignée, mais aussi qu'il puisse se coordonner avec d'autres agents s'il faut gérer des dépendances entre les sous-tâches ou s'il a besoin de fonctions assurées par d'autres agents.

Ainsi pour répondre à la consigne, qui était, construire un système SMA permettant de simuler le comportement d'une population par simulation des individus qui la compose.

Nous avons présentés dans ce rapport nos travaux sur la conception d'un système multi-agent modélisant un trafic routier.

Dans la réalité, les voitures se partagent la route, il y a des interactions entre elles. Le Code de la route prévoit des règles permettant aux conducteurs de partager l'espace routier aux intersections en toute sécurité.
Ainsi, le conducteur doit apprendre à se positionner sur la chaussée, respecter les feux rouges, les stop et encore à s'adapter à des flux de circulation particulièrement denses.

Pour pouvoir créer ce modele on a suivis la décomposition d'un SMA en trois dimensions :
Dans un premier temps nous avons étudier les agents en définissant les composants du système.
Nous avons ensuite crée l'environnement, milieu dans lequel sont plongés les agents, composé d'objets qui sont perçus et manipulés par les agents.
Et finalement il a fallut mettre en place des interactions, soit l'ensemble des infrastructures, langages et protocoles d'interaction entres les agents.

II/ ANALYSE DU PROBLÈME

1. SEMAINE 1.2.3: ETUDE DU SUJET + DÉFINITION DES AGENTS

1.1. Étudier le sujet

Pendant les premières semaines, nous avons dû prendre le temps d'étudier le sujet et de trouver **notre thème.**
Par conséquent, on s'est posé des questions concernant la création d'un réseau routier.
Quel agent pourrait être intéressant ? Quel est la nature de l'agent ? Quel type de route ? Des feux ? Des passages piétons ?

1.2. Comprendre le fonctionnement d'un agent

Selon la définition générale, un agent est une entité autonome, physique ou virtuelle, capable de percevoir partiellement son environnement et d'agir sur ce même environnement [Ferber 1995].

L'environnement d'un agent est tout ce qui est extérieur à l'agent, c'est-à-dire les autres agents, les objets, les variables globales, les lois et les moyens techniques permettant la simulation.

La structure d'un agent est la suivante :
- **Compétences** : ce qu'il peut faire, son rôle au sein du système.
- **Connaissances** : connaissances partielles sur lui-même et sur son environnement.
- **Accointances** : relations avec d'autres agents.
- **Aptitudes** : capacités de perception, de raisonnement, de décision et d'action.

Le comportement d'un agent est souvent représenté par le cycle de vie : Perception, Décision, Action.
L'agent perçoit son environnement lors de la phase de perception. Il décide alors des actions qu'il va effectuer en fonction de ce qu'il a perçu et de ses connaissances. L'agent termine son cycle en réalisant les actions qu'il a choisies lors de la phase de décision.

2. SEMAINE 4/5: ETUDE DES INTERFACES GRAPHIQUES

2.1. Découverte des interfaces graphiques

Dans cette partie, nous étudions **les interfaces graphiques.**
Le langage Java propose différentes bibliothèques d'interface graphique, mais dans notre projet, nous utiliserons les packages **javax.swing et java.awt** présents d'office dans Java.

Ces premières semaines nous ont permis d'apprendre à utiliser l'objet **JFrame**, présent dans le package java.swing. Cela nous a permis de manipuler la création d'une fenêtre, définir sa taille, etc.
Une fenêtre n'est qu'une multitude de composants posés les uns sur les autres et que chacun possède un rôle qui lui est propre.

2.2. Construction de notre première fenêtre avec Jframe

JFrame, tout comme JPanel, est également une sous-classe de JComponent et JContainer.
C'est une fenêtre avec ses propres caractéristiques. Il a une bordure, une barre de titre etc.
Ses attributs physiques, comme la taille, la couleur, les polices, etc., peuvent tous être personnalisés.

Pour commencer nous avons créer une classe nommé **Fenêtre**.
- Hérite de JFrame
- Constructeur dans lequel nous plaçons nos instructions.

```
public Fenêtre() {
    this.setTitle("Circulation routière");
    this.setSize(1500, 950);
    this.setLocationRelativeTo((Component)null);
    this.setDefaultCloseOperation(3);
    this.setContentPane(this.pan);
    this.setVisible(true);
```

Pour cela on définit un titre pour notre fenêtre « Circulation routière », on définit sa taille qui est de 1500 pixels de large et de 950 pixels de haut; On demande a notre objet de se positionner au centre ; Termine le processus lorsqu'on appuie sur la croix rouge.

Par défaut, une JFrame est toujours invisible, c'est-à-dire que la fenêtre est créée mais jamais affichée par défaut. C'est pour cela que l'on rajoute la ligne f.setVisible(true); qui permet de rendre visible la fenêtre.

III/ MODELISATION

1.SEMAINE 6: CLASSE + HÉRITAGE

1.1. Classes choisies et lien entre elles

Apres l'étude de notre sujet, nous sommes arrivées a la conclusion que nous avons besoins de deux types d'agent : des agents mobiles et des agents immobiles.

Les agents mobiles sont des agents dont les positions seront modifiées au cours du temps. Pour notre projet, on distingue des agents mobiles de type voitures, bus, motos…

Les agents immobiles quant à eux auront des positions invariables on distingue parmi eux les feux et les routes.

Néanmoins, on constate que ces deux classes possèdes de nombreuses attributs en commun. Par conséquent, on a crée la classe Agent qui sera une super-classe des classes AgentMobile et AgentImmobile.

1.2. Code de ces classes

AGENT :

- Class abstraite
- Attributs : int posX, int posY, int largeur, int hauteur, Color couleur
- Constructeur qui permet d'instancer tous les attributs
- Getters/Setters
- Méthode toString pour l'affichage

AGENT MOBILE

- Hérite de la class Agent
- Attribut : `TypeAgent type , int vitesse`
- Constructeur qui permet d'instancier les attributs de la class Agent + les attributs de la class AgentMobile
- Getters/Setters

Nous avons aussi créer des méthodes pour déplacer nos différents agents. Pour cela, on part du principe que lors du démarrage la vitesse de la voiture est égale 0. Pour atteindre sa vitesse de croisière, la vitesse augmente au fur et a mesure.

La vitesse est mesuré par la distance parcouru par rapport au temps, ainsi **dans notre projet la vitesse sera représenté par le nombre de pixels parcouru en un temps t** qui correspond à la durée de suspension du thread (chez nous ce temps sera égalé à 50millis)

-Méthode `avanceX()` :

- Résultat : void donc ne retourne rien
- Rôle : attribut une nouvelle position x à l'agent
- Utilisation : lorsqu'on souhaite déplacer un agent mobile vers la droite
- Explication du code : On crée un boucle for qui va augmenter la vitesse d'une unité jusqu'a atteindre la vitesse de l'agent, instanciée lors de la création de celui ci. Mathématiquement, c'est une suite arithmétique de raison 1 jusqu'à atteindre sa vitesse de croisière.

{Un+1 = Un + r ; 0<= n <vitesse}
{Un = vitesse ; n = vitesse }

-Méthode `reculerX()` :

- Résultat : void donc ne retourne rien
- Rôle : attribut une nouvelle position x à l'agent
- Utilisation : lorsqu'on souhaite déplacer un agent mobile vers la gauche

- Explication du code : même principe que la méthode `avanceX()`

-`Méthode accelerationX()`:
- Résultat : void donc ne retourne rien
- Rôle : attribut une nouvelle position x à l'agent
- Utilisation : lorsqu'on souhaite qu'un agent mobile se déplace plus rapidement vers la gauche.

- Explication: On part du principe que l'accélération permet d'augmenter de façon considérable la position par rapport au temps. Dans notre projet, l'accélération permettra donc d'atteindre plus rapidement la vitesse par unité de 3.

-Méthode `ralentirX()`:
- Résultat : void donc ne retourne rien
- Rôle : attribut une nouvelle position x à l'agent
- Utilisation : lorsqu'on souhaite qu'un agent mobile décéléré à l'horizontale

-Méthode `avancerY()`:
- Résultat : void donc ne retourne rien
- Rôle : attribut une nouvelle position y à l'agent
- Utilisation : lorsqu'on souhaite déplacer un agent mobile vers le bas
- Explication du code : même principe que la méthode `avanceX()`

-Méthode `reculerY()`:
- Résultat : void donc ne retourne rien
- Rôle : attribut une nouvelle position y à l'agent
- Utilisation : lorsqu'on souhaite déplacer un agent mobile vers le haut
- Explication du code : même principe que la méthode `avanceX()`

-Méthode `accelerationY()`:
- Résultat : void donc ne retourne rien
- Rôle : attribut une nouvelle position y à l'agent
- Utilisation : lorsqu'on souhaite qu'un agent mobile se déplace plus rapidement vers le bas.
- Explication: même principe que la méthode `accelerationX()`

-Méthode `ralentirY()`:
- Résultat : void donc ne retourne rien
- Rôle : attribut une nouvelle position y à l'agent
- Utilisation : lorsqu'on souhaite qu'un agent mobile décéléré à la verticale

-Méthode `repos()`:
- Résultat : void donc ne retourne rien
- Rôle : maintient l'agent à l'arrêt
- Utilisation : lorsqu'on souhaite qu'un agent mobile s'arrête

AGENTS IMMOBILES :
- Hérite de la class Agent
- Class abstraite
- Attribut : int numero
- Constructeur qui permet d'instancier les attributs de la class Agent + int numero

- Getters/Setters

FEU1 :
Les feux seront placé à une interaction entre deux routes, après une durée déterminé ils changent de couleur.

- Hérite de la class AgentImmobile
- Constructeur qui permet d'instancier les attributs de la super classe AgentImmobile

-Méthode `changementDeCouleur()`:
- Résultat : void donc ne retourne rien
- Rôle : lorsqu'on appelle la fonction si l'objet est rouge alors il revient noir, sinon il devient rouge
- Utilisation : lorsqu'on souhaite qu'un feu passe au rouge

-Méthode `FeuVert()`:
- Résultat : void donc ne retourne rien
- Rôle : lorsqu'on appelle la fonction si l'objet est noir alors il revient vert, sinon il devient noir
- Utilisation : lorsqu'on souhaite qu'un feu passe au vert

-Méthode changeDeCouleurOrange():

ROUTE :
La route est la variable qui représentera les voies où circulent les voitures dans le trafic.
-Hérite de la class AgentImmobile
-Constructeur qui permet d'instancier les attributs de la class AgentImmobile

TYPE AGENT :
Une énumération est en fait une classe, d'où cette appellation de classe énumération. La classe TypeAgent étend la classe `Enum`.
Les valeurs d'une énumération sont les seules instances possibles de cette classe. Dans notre projet, `TypeAgent` comporte sept instances seulement : `Voiture`, `pompiers`, `police`, `Bus`, `taxi`, `personne`, `cycliste`, `ambulance`, `divers`.
On peut donc comparer ces instances à l'aide d'un == de façon sûre, même si la comparaison à l'aide de la méthode `equals()` reste possible.

2. SEMAINE 7.8: ENVIRONNEMENT AVEC JPANEL + GESTION DES IMAGES

2.1. Création d'un panneau pour gérer les placements

Nous traiterons de `java.swing` et de `java.awt` . Nous n'utiliserons pas de composants `awt`, nous travaillerons uniquement avec des composants `swing` ; en revanche, des objets issus du package `awt` seront utilisés afin d'interagir et de communiquer avec les composants `swing`.

Nous avons créer une classe panneau qui hérite de JPanel.

- Public void paintComponent(Graphics g) :
Cette méthode est celle que l'objet appelle pour se dessiner sur notre fenêtre. Elle permet aussi de s'assurer que le panneau sera dessiné comme n'importe quel `JPanel`.

On obtiendra des rectangles qui permettent de dessiner des trottoirs de largeur × hauteur = 1500x220 et d'autres de largeur x hauteur = 220x950

Ll27/L36 : Lecture de deux fichiers images utilisés en fond de largeur 1500 et de hauteur 950.

L38/L42 : Création de nos trottoirs par des rectangles de hauteur 1500x220 et de 950x220 avec g2d.setPaint et g2d.fillRect().

L45/L50 : Boucle for qui affiche un rectangle pour chaque agent Route, on a utiliser les méthodes getPosX, getPosY, getLargeur, getHauteur pour listeRoute

L50/L151 : Création de rectangle blanc pour les passages piétons (largeur 50, hauteur 5) et les séparations blanches sur la route (de largeur 20 hauteur 40) et inversement

L155/L170 : Lecture du fichier image du panneau indiquant un passage piéton et le place en fonction de x et y, de largeur 30 et de hauteur 30.

L176/201 : Lire le fichier image de notre panneau feu et la place les feux avec g.drawImage() en fonction de x et y, de largeur 50 et de hauteur 100.

L203/235 : Affichage d'une image différente selon l'instanciation du type de l'agent pour listeMobile1

L238/272 : Affichage d'une image différente selon l'instanciation du type de l'agent pour listeMobile2

L273/303 : Affichage d'une image différente selon l'instanciation du type de l'agent pour listeMobile3

L304/334 : Affichage d'une image différente selon l'instanciation du type de l'agent pour listeMobile4

L335/407 : Pareil pour listeMobile5, listeMobile6, listeMobile7 et listeMobile8

L410/427 : Création d'un oval pour les feux vert, rouge, orange.

2.2. Etude de l'insertion des images et construction en 2D

Utilisation de la classe ImageIO(), afin d'ajouter une image au JPanel. Cette classe possède entre autres la méthode read() permettant de récupérer le contenu de l'**image** à partir du fichier passé en paramètre.
Ces méthodes permettent de charger des images à partir de fichiers JPG, PNG et GIF et l'affichage des images a l'aide de la méthode drawImage. La méthode fill est utilisé pour la création des lignes, rectangles, ovales et polygones de Graphics2D.

Suite de la classe Fenetre :

- Hérite de JFrame

- Création d'un objet Panneau qui sert de cadre initialement invisible sans fenêtre propre utile pour ordonner les contrôles.
- Création de 4 attributs static Route en fonction de la position x et y et de largeur 1500 hauteur 150 pour les routes horizontales et de largeur 150 hauteur 950 pour les routes verticales.

- Création d’une listeRoute d’objet Route où l'on ajoute les quatre routes créée précédemment.

- Création objets static Feu1
- Création d’une listeFeu d’objet Feu1 constitué de feux rouges qui gèrent la circulation a l’horizontale.
- Création d’une listeFeuV d’objet Feu1 constitué de feux verts qui gèrent la circulation a l’horizontale.
- Création d’une listeFeu2 d’objet Feu1 constitué de feux rouges qui gèrent la circulation a la verticale.
- Création d’une listeFeuV2 d’objet Feu1 constitué de feux verts qui gèrent la circulation a la verticale.

- Création objets static AgentMobile en fonction de la position x et y , on lui attribut une largeur, une hauteur, une vitesse et un type.
- Création d’une listeMobile1 : circulation horizontale des agents qui roule vers la droite sur la route du haut
- Création d’une listeMobile2 : : circulation horizontale des agents qui roule vers la droite sur la route du bas
- Création d’une listeMobile3 : circulation horizontale des agents qui roule vers la gauche sur la route du haut
- Création d’une listeMobile4 : circulation horizontale des agents qui roule vers la gauche sur la route du bas
-
- Création d’une listeMobile5 : circulation verticale des agents qui roulent vers le bas sur la route de gauche
- Création d’une listeMobile6 : circulation verticale des agents qui roulent vers le haut sur la route de gauche
- Création d’une listeMobile7 : circulation verticale des agents qui roulent vers le bas sur la route de droite
- Création d’une listeMobile8 : circulation verticale des agents qui roulent vers le haut sur la route de droite

Public ThreadMobile1
- Initialisation et mise en marche du thread pour la listeMobile1

Même principe pour toutes les ThreadMobile avec leur listeMobile respective.

Public ThreadFeu
- Initialisation et mise en marche du thread pour la listeFeu

Public class EssaieMobile1 : permet d'exécuter le thread par l'implémentation de la méthode runnable et aussi fait marcher le thread pour l’agent lui étant associé.
C’est le cas pour tous les Threads EssaiMobile.

- Private void Mobile1( ) :
Création d’une boucle for qui appelle la méthode circulationHorizontale() pour tous les agents de la liste courante.
This.pan.repaint() pour redessiner les nouvelles positions des agents de la liste courante après déplacement.
Thread.sleep entraîne la suspension de l'exécution du thread pendant une période spécifiée.
Création d’une boucle for pour régénérer les agents de la liste en fin de parcours.
C’est le cas pour toutes les fonctions Mobile de la Fenetre.

IV/ APPROFONDISSEMENT

1. SEMAINE 7.8.9 : ETUDE DES COMPORTEMENTS ENTRE LES AGENTS

1.1. Comportement des agents qui se déplacent à l'horizontal

On rappelle que nous avons un ensemble d'agents qui interagissent dans un environnement commun et qui participent à une dynamique d'organisation.
Ainsi, on a du organiser nos agents de façon a ne pas rentrer en collision et ainsi ne pas avoir des accidents.
Dans la classe AgentMobile, on a codé des méthodes qui gèrent le comportement entre les agents.

`-Méthode champsDeVisonX(listeMobile[] l1):`

- Résultat : entier
- Rôle : retourne la distance minimum entre mon agent et un agent extérieur qui se trouvent dans la même liste(soit sur la route qui roule dans la même direction
- Utilisation : L'agent mobile recherche si un ou plusieurs agents se trouvent devant lui (posX >) Si oui lequel est le plus proche de lui ?

- Explication du code :

_ Création d'une ArrayList<> `distances`
_ Création d'une boucle qui calcule `a` (= distance x entre un agent i et les autres agents de la listeMobile[] l entrés en paramètre)
_Si `a` > 0 : ajouter `a` à la liste
_Création d'une variable `minimum` de type int = 200
_Si la taille de la liste `distance` est >0 (ce qui signifie qu'il y a des voitures devant l'agent)
_Alors on attribut la valeur minimum de la liste distance à la notre variable minimum.

_Sinon le `minimum` reste à 200 (ainsi le comportement agent ne sera pas influencer par un autre agent)

`-Méthode circulationHorizontale1(FEU1 f, listeMobile[] l1, listeMobile[] l2, listeMobile[] l3, listeMobile[] l4):`

- Résultat : void donc ne retourne rien
- Rôle : permet de mettre en mouvement les voitures horizontales en prenant en compte la gestion des cohésions et des visions des agents.
- Utilisation : lorsqu'on souhaite faire circuler les agents horizontalement.
- Explication du code :

La fonction circulationHorziontale1() : prends en paramètres un feu et 4 listes, la première liste sera modifiée en fonction de la position des éléments appartenant aux 3 autres listes, qui constituent l'environnement.

_Création d'un entier minimum qui fait appelle a la fonction champsDeVision qui prend en paramètre la liste de voiture que l'on veut déplacer

_Mise en place d'un indicateur qui nous informe si un obstacle(AgentMobile) est présent ou non sur la trajectoire future de l'agent. (Int valeur = 0)

Comment se comporte l'agent quand le feu est vert ? Quand le feu est-il rouge ? Quand il se trouve trop prêt d'un autre agent ?

_Si le feu est vert, soit le feu f en paramètre est == Color.green :

_**Si il n'y a aucun obstacle** (indicateur = 0) sur la trajectoire que va prendre l'agent, il peut donc circuler.

_ Si le minimum est entre 0 et 70 : Alors appelle de la fonction repos() pour arrêter l'agent

_ Si le minimum est entre 70 et >200 : Alors appelle de la fonction ralentirX() pour ralentir la vitesse de l'agent

_Sinon appelle de la fonction avanceX()

Si il y'a un obstacle sur la trajectoire que va prendre l'agent, alors il s'arrête et reste au repos en attendant que l'obstacle disparaisse de sa trajectoire, a l'aide de la méthode repos().

_Sinon (le feu est rouge) :

***Création d'une variable PosI = getPosX() pour avoir la position x**

_Si PosI est entre 0 et 150 (si la voiture se trouve avant le passage piéton du premier feu)

_ Si le minimum est entre 0 et 70 : Alors appelle de la fonction repo() pour arrêter l'agent
(Si il y a autre agent devant a une distance entre 0 et 70 , alors mon agent doit s'arrêter)

_ Si le minimum est entre 70 et >200 : Alors appelle de la fonction ralentirX() pour ralentir la vitesse de l'agent
(Si il y a autre agent devant a une distance entre 70 et 200 , alors mon agent doit ralentir)

_Sinon appelle de la fonction avanceX() (Sinon il peut continuer jusqu'au feu)

_Si PosI est entre 170 et 850 **(si la voiture se trouve après le premier feu et avant le passage piéton du deuxième feu)**

_ Si le minimum est entre 0 et 70 : Alors appelle de la fonction repos() pour arrêter l'agent

_ Si le minimum est entre 70 et >200 : Alors appelle de la fonction ralentirX() pour ralentir la vitesse de l'agent

_Sinon appelle de la fonction avanceX()

_Si PosI est entre 850 et 950

_Appelle de la fonction repos() pour arrêter l'agent

_Si PosI est entre 950 et 1500

_Appelle de la fonction accelerationX() pour l'agent se déplace plus vite

_Sinon

_Appelle de la fonction repos() pour arrêter l'agent

`-Méthode circulationHorizontale(FEU1 f, listeMobile[] l1, listeMobile[] l2, listeMobile[] l3, listeMobile[] l4):`

1.2. Comportement des agents qui se déplacent à la verticale

`-Méthode champsDeVisonX(listeMobile[] l1):`

- Résultat : entier
- Rôle : retourne la distance minimum entre mon agent et un agent extérieur qui se trouvent dans la même liste (soit sur la route qui roule dans la même direction)
- Utilisation : L'agent mobile recherche si un ou plusieurs agents se trouvent devant lui (posY <) Si oui lequel est le plus proche de lui ?

- Explication du code :

_ Création d'une ArrayList<> `distances`
_ Création d'une boucle qui calcule `a` (= distance y entre un agent i et les autres agents de la listeMobile[] l entrés en paramètre)
_Si `a < 0` : ajouter `a` à la liste
_Création d'une variable `minimum` de type int = 200
_Si la taille de la liste `distance` est >0 (ce qui signifie qu'il y a des voitures devant l'agent)
_Alors on attribut la valeur minimum de la liste distance à la notre variable minimum.

_Sinon le `minimum` reste à 200 (ainsi le comportement agent ne sera pas influencer par un autre agent)

```
-Méthode circulationVerticale1(FEU1 f, listeMobile[] l1, listeMobile[]
l2, listeMobile[] l3, listeMobile[] l4):

-Méthode circulationVerticale2(FEU1 f, listeMobile[] l1, listeMobile[]
l2, listeMobile[] l3, listeMobile[] l4):

-Méthode circulationVerticale3(FEU1 f, listeMobile[] l1, listeMobile[]
l2, listeMobile[] l3, listeMobile[] l4):

-Méthode circulationVerticale4(FEU1 f, listeMobile[] l1, listeMobile[]
l2, listeMobile[] l3, listeMobile[] l4):
```

- Résultat : void donc ne retourne rien
- Rôle : permet de mettre en mouvement les voitures verticales en prenant en compte la gestion des cohésions et des visions des agents.
- Utilisation : lorsqu'on souhaite faire circuler les agents verticalement.
- Explication du code : même principe que -Méthode circulationHorizontale1()

V/ BILAN

A travers la première partie et grâce au notion de création de variable apprises en cours on a pu créer les différents agents mobiles et immobiles que nous utiliserons dans notre projets.

Nos investigations et les animations faites lors dés TP nous ont permis de pouvoir réaliser à partir de Swing notre Panneau et notre Fenêtre, nous permettant ainsi de faire une première animation mobile de nos variables.

Dans la suite de notre projet, nous avons améliorer les comportements de nos variables à partir des notions d'héritage, d'interfaces et de classes abstraites puis nous avons ajouter types de véhicules.

En ce qui concerne la partie animation, nous chercherons à adapter au mieux notre fenêtre à la réalité.

Rapport individuel :
Aya

Semaine 1.2.3 :
- Découverte du sujet et bien comprendre les concepts .
- Choix du thème
- Etude des interfaces graphique

Semaine 4.5 :
Création de la classe agentMobile qui hérite de la classe agent. Dans notre projet, les agents mobiles seront des agents dont les positions seront modifiées au cours du temps. On distingue différents type d'agent : pour les gérer j'ai créer la classe TypeAgent qui hérite de la classe enum et qui comporte les instances dont j'aurais besoin.

La classe agentMobile a beaucoup été modifié au long du projet car on cherchait à avoir une représentation proche des comportement des voitures dans la réalité.

Ainsi, j'ai crée des méthodes utiles pour que les agentsMobiles puissent se déplacer.

Pour cela, on part du principe que lors du démarrage la vitesse de la voiture est égale 0. Pour atteindre sa vitesse de croisière, la vitesse augmente au fur et a mesure.

La vitesse est mesuré par la distance parcouru par rapport au temps, ainsi **dans notre projet la vitesse sera représenté par le nombre de pixels parcouru en un temps t** qui correspond à la durée de suspension du thread (chez nous ce temps sera égalé à 50millis)

Explication du code : On crée un boucle for qui va augmenter la vitesse d'une unité jusqu'a atteindre la vitesse de l'agent, instanciée lors de la création de celui ci. Mathématiquement, c'est une suite arithmétique de raison 1 jusqu'à atteindre sa vitesse de croisière.

{Un+1 = Un + r ; 0<= n <vitesse}
{Un = vitesse ; n = vitesse }

Semaine 6.7 :
_Construction des agents mobiles dans le panneau.

Dans le panneau, j'ai fais la construction des agents mobiles.

L203/407 : Affichage d'une image différentes selon l'instanciation du type de l'agent pour listeMobile1 jusqu'à listeMobile8. Les images sont différentes pour chaque type et selon les listeMobile.
Place les images dans des blocs try{}cacth{} pour gérer les exceptions. Dans le try, une boucle for qui selon le type de l'agent mobile, lit un fichier image et place l'image avec g.drawImage() en fonction de x et y, de la largeur et hauteur.

Semaine 8.9 :
Création des méthodes champsDeVisionX() et circuationHorizontale() dans agentMobile :

La partie la plus compliqué pour moi a été l'ecriture de la fonction circulationHorizontale() car en effet il est quand même très compliqué de prévoir la réaction d'un conducteur dans tout les cas.

Avant de réagir il a fallut que notre agent mobile puisse percevoir son entourage. Ainsi j'ai crée la méthode chamsDeVisionX(), cette méthode fait une liste de tout les agents qui se trouve à une distance x > 0 de cette agent. Ensuite, on recherche le minimum de cette liste soit la distance la plus petite, ainsi la distance entre notre agent et l'agent qui se trouve le plus proche devant elle. Le but de cette méthode est donc de nous renvoyer la distance la petite qu'il y a entre mon agent et l'environnement.

Ensuite, j'ai créer la méthode circulationHorizontale() : le code est détaillée plus haut dans le rapport.
Mais le principe est que le déplacement d'un agent est influencée par la couleur du feu qui se trouvent sur sa route, la voiture qui roulent devant elle et les voitures qui coupent sa routent a la verticale.

Rapport individuel :
Guichel

Semaine 1.2.3 :

- Découverte du sujet et bien comprendre les concepts .
- Choix du thème
- Etude des interfaces graphique

Semaine 4.5 :
Création de la première fenêtre et premier panneau

Faisant partie des évolutions apportés par J2SE par rapport aux versions anciennes,Swing est une forme de bibliothèque graphique incorporé dans le package Foundation classe.Durant cette semaine j'ai personnellement travaillé sur les objets swing et awt super important dans l'affichage de nos agents .Le JFrame et le JPanel sont les classes swing qui nous ont permis de construire respectivement la fenêtre principale et le panneau de notre projet

Héritière de la classe JFRAME , la fenêtre que j'ai crée possède plusieurs méthodes, threads ,variables.C'est à travers elles que les agents sont animé chacun selon un thread que j'ai bien définie dans la fenêtre.La fenetre est comme un atelier ou tout les element sont mis en fonctionnement elle possède des dimensions, une couleurs bien definies qui construire ainsi l'écran d'affichage.

La classe panneau quant à elle est une fille de la classe Jpanel , elle comprend la méthode paintCompennt que
que je considère comme le chantier de notre projet c'est là ou se trouve toute les constructions des formes présentes sur l'animation(route, passages pietons,feux,...).

Semaine 6.7 :
Gestion des images

A travers le titre gestion des images, je veux parler des images présentes sur l'affichage. Durant les semaines 6 et 7 j'ai travaillé sur les images en java et leurs porté dans notre projet, Pour cela j'ai du importer l'objet image, io.image tout comme le file pour lire les chemin des images de notre projet.J'ai place les images dans des blocs try{}cacth{} pour gere les exceptions.Tout en calibrant bien leurs dimensions

Construction
Dans cette partie je travaille avec des formes géométrique de base pour construire de route, des passages piéton,...Puis je suis passé à la construction 2d des couleurs composées, des trottoirs présente dans l'animation

Semaine 8.9 :
Après avoir travaillé et étudié l'affichage de l'animation , je suis passé dans la constructions des méthodes circulations des mobiles verticaux et de la méthodes champsVision().Étant les

dernières grandes méthodes à réaliser, ces fonctions résument l'ensemble des comportement que l'agent vertical peut adopter par rapport à son environnement(feu, véhicules,..);Ces méthode prennent en paramètre tout les obstacles présentes sur sa trajectoire et agit en fonction du feu et de ces obstacles

les Threads sont des fils d'exécution des fonctions présentes dans la fenetre. Bien entendu à la fin de la conception de ces méthodes, j'ai construire des threads dans la fenêtre pour mettre en circulation les agents mobiles verticaux

En somme , ce projet m'a permis de pouvoir approfondir mes connaissances et de comprendre un peu plus ce langage je tiens à remercier les membres de mon groupe et les profs pour ce projet.

Rapport individuel :
Mehdi

Semaine 1.2.3 :

- Découverte du sujet et bien comprendre les concepts .
- Choix du thème
- Etude des interfaces graphique

Ma première semaine sur ce projet était principalement consacrée aux recherches sur le concept du système multi-agents.
On parle de simulation entre agents dans une structure afin de choisir un thème adéquat que mon groupe et moi pourrait concrétiser. J'ai commencé par répertorier ce qui était nécessaire dans un système comme celui-ci, c'est-à-dire : l'environnement partagé entre agents, l'interaction entre les agents et les coordinations d'actions entre eux. Mon groupe et moi avons alors opté pour la représentation d'un trafic routier en tant que système multi-agents.
J'ai d'abord cherché à réaliser une interface graphique, en me documentant et en travaillant sur les objets Swing et awt. JFrame était l'idéal pour la création de l'interface, c'est une classe qui se trouve dans le package javax.swing et qui hérite de java.awt.frame, plus précisément il s'agit d'une fenêtre avec bordures et une barre pour mentionner le titre de notre simulation.

Semaine 4.5 :
Création de la classe Agent Immobile, Feu1 et Route
A la quatrième semaine, après avoir assimilé le concept et les outils dont j'allais avoir besoin, j'ai commencé par créer la classe AgentImmobile avec les attributs suivants : posX, posY, largeur, hauteur, Color couleur (en important java.awt.Color) et numero. C'est une classe abstraite qui hérite de la super-classe Agent et qui comme son nom l'indique contient deux classes qui seront nos deux agents immobiles : les feux, et les routes. Cette classe dispose d'un constructeur qui permet d'instancier les attributs de la classe et possède un attribut de plus que la super-classe, le numéro (int numéro) un entier qui permet de repérer les routes et les feux.
La classe Route hérite de la classe AgentImmobile et représente les quatre routes où circulent les voitures dans le trafic.
La classe Feu1 hérite également de la classe AgentImmobile et possède trois méthodes pour les feux tricolores qu'on utilise, les méthodes sont changeDeCouleur(), FeuVert(), changeDeCouleurOrange() et permettent de changer les trois couleurs des cercles du feu tricolore du rouge au noir au rouge, du vert au noir au vert et de l'orange au noir à l'orange
.
Semaine 6.7 :
Construction des routes et des feux sur le panneau
Entre la cinquième et la sixième semaine, je me suis occupé de la construction des feux et des routes dans la classe panneau qui hérite de JPanel qui permet de s'assurer et de réaliser le dessin du panneau de notre projet, en utilisant quelques méthodes de la classe Graphics telles que :

fillRect ou encore fillOval pour dessiner des lignes, des rectangles et des cercles pour la construction des passages piétons, des lignes discontinues sur nos routes et pour les cercles représentant les feux sur les feux tricolores, en ajustant au mieux les positions x et y ainsi que la largeur et la hauteur pour les éléments.

Semaine 8.9 :
Création des Threads pour les agents immobiles
Vers les dernières semaines, je me suis concentré sur les Threads qui permettent la construction des agents immobiles sur l'animation. Les Threads sont des blocs d'instructions qui ont la particularité d'être exécutés simultanément, partant de ce principe il est donc paru nécessaire pour mon groupe d'utiliser les Threads pour construire les agents. Par conséquent, dans la Fenetre j'ai créé les Threads nécessaires pour réaliser les agents immobiles dont on avait besoin. Personnellement, ce projet a été décisif, car il m'a permis d'apprendre et de développer plusieurs notions en Java et ainsi me doter de compétences solides pour la suite de mes projets, le travail collectif et la présence des professeurs a été important pour gérer au mieux la gestion des cohésions et la gestion des visions des agents afin de représenter au mieux la réalité dans notre projet.