Simulation de ville

Nous allons créer une **simulation de ville** où nous modéliserons les **voitures**, les **piétons**, les **bâtiments**, et des **règles de circulation**. L'objectif est d'établir une base simple pour simuler la vie urbaine. Voici le plan du projet :

Plan de la simulation

1. **Modélisation des Classes**
    - Classe **Voiture**
    - Classe **Piéton**
    - Classe **Bâtiment**
    - Classe **Ville**
    - Gestion des **règles de circulation**
2. **Interactions entre les objets**
    - Circulation des voitures et des piétons dans la ville
    - Interaction avec les bâtiments
3. **Simulation avec boucle de temps**
    - Lancer la simulation et observer les interactions dans le temps

1. Modélisation des Classes

Classe Voiture

La classe `Voiture` va modéliser des véhicules dans la ville. Chaque voiture a une marque, une vitesse maximale, une position, et la capacité de se déplacer.

```
class Voiture:
    def __init__(self, marque, vitesse_max):
        self.marque = marque
        self.vitesse_max = vitesse_max
        self.vitesse_actuelle = 0
        self.position = 0  # Position sur une route simple (ex: 1D)

    def accelerer(self, valeur):
        self.vitesse_actuelle += valeur
        if self.vitesse_actuelle > self.vitesse_max:
            self.vitesse_actuelle = self.vitesse_max
        print(f"{self.marque} accélère à {self.vitesse_actuelle} km/h.")

    def freiner(self, valeur):
        self.vitesse_actuelle -= valeur
        if self.vitesse_actuelle < 0:
            self.vitesse_actuelle = 0
        print(f"{self.marque} ralentit à {self.vitesse_actuelle} km/h.")

    def avancer(self):
        self.position += self.vitesse_actuelle
        print(f"{self.marque} se déplace à la position {self.position}.")
```

Classe Piéton

Un piéton peut marcher dans la ville avec une vitesse de marche et une position.

```
class Pieton:
    def __init__(self, nom, vitesse_marche):
        self.nom = nom
        self.vitesse_marche = vitesse_marche
        self.position = 0

    def marcher(self):
        self.position += self.vitesse_marche
        print(f"{self.nom} marche à la position {self.position}.")
```

Classe Bâtiment

Les bâtiments dans la ville peuvent représenter des maisons, des bureaux ou des magasins. Ils ont une adresse (position dans la ville).

```
class Batiment:
    def __init__(self, nom, adresse):
        self.nom = nom
        self.adresse = adresse

    def afficher_info(self):
        print(f"{self.nom} est situé à l'adresse {self.adresse}.")
```

Classe Ville

La ville va contenir les voitures, les piétons et les bâtiments. Elle va gérer les interactions et la simulation générale.

```
class Ville:
    def __init__(self, nom):
        self.nom = nom
        self.voitures = []
        self.pietons = []
        self.batiments = []

    def ajouter_voiture(self, voiture):
        self.voitures.append(voiture)
        print(f"Voiture ajoutée : {voiture.marque}")

    def ajouter_pieton(self, pieton):
        self.pietons.append(pieton)
        print(f"Piéton ajouté : {pieton.nom}")

    def ajouter_batiment(self, batiment):
        self.batiments.append(batiment)
        print(f"Bâtiment ajouté : {batiment.nom}")

    def simuler(self, temps):
        for _ in range(temps):
            print("\n-- Simulation --")
            for voiture in self.voitures:
                voiture.avancer()
            for pieton in self.pietons:
                pieton.marcher()
```

2. Interactions entre les objets

Règles de circulation basiques

Dans la simulation, nous allons définir des règles simples :

- Les voitures roulent sur une route droite.
- Les piétons marchent sur le trottoir.
- Si une voiture ou un piéton atteint un bâtiment, elle/il s'arrête.

Modification des classes Voiture et Piéton pour inclure l'interaction avec les bâtiments

```
class Voiture:
    def __init__(self, marque, vitesse_max, ville):
        self.marque = marque
        self.vitesse_max = vitesse_max
        self.vitesse_actuelle = 0
        self.position = 0
        self.ville = ville

    def avancer(self):
        self.position += self.vitesse_actuelle
        print(f"{self.marque} se déplace à la position {self.position}.")
        self.verifier_batiment()

    def verifier_batiment(self):
        for batiment in self.ville.batiments:
            if self.position >= batiment.adresse:
                print(f"{self.marque} est arrivé(e) au {batiment.nom} et
s'arrête.")
                self.vitesse_actuelle = 0

class Pieton:
    def __init__(self, nom, vitesse_marche, ville):
        self.nom = nom
        self.vitesse_marche = vitesse_marche
        self.position = 0
        self.ville = ville

    def marcher(self):
        self.position += self.vitesse_marche
        print(f"{self.nom} marche à la position {self.position}.")
        self.verifier_batiment()

    def verifier_batiment(self):
        for batiment in self.ville.batiments:
            if self.position >= batiment.adresse:
                print(f"{self.nom} est arrivé(e) au {batiment.nom} et
s'arrête.")
                self.vitesse_marche = 0
```

3. Simulation avec boucle de temps

Nous allons maintenant **lancer une simulation** où des voitures et des piétons se déplacent dans la ville, en tenant compte des bâtiments.

Création d'une ville et de ses éléments

```
# Création de la ville
ville = Ville("Paris")

# Ajout de bâtiments
batiment1 = Batiment("Maison", 50)
batiment2 = Batiment("Bureau", 100)
ville.ajouter_batiment(batiment1)
ville.ajouter_batiment(batiment2)

# Ajout de voitures
voiture1 = Voiture("Tesla", 50, ville)
voiture2 = Voiture("BMW", 60, ville)
ville.ajouter_voiture(voiture1)
ville.ajouter_voiture(voiture2)

# Ajout de piétons
pieton1 = Pieton("Alice", 5, ville)
pieton2 = Pieton("Bob", 7, ville)
ville.ajouter_pieton(pieton1)
ville.ajouter_pieton(pieton2)

# Simulation pendant 10 unités de temps
ville.simuler(10)
```

Améliorations possibles

1. **Feux de circulation :** Implémenter des feux rouges et verts pour les voitures et les piétons.
2. **Réseau de routes :** Étendre la simulation à un réseau de routes en 2D pour rendre les déplacements plus complexes.
3. **Ajout de collisions :** Gérer les collisions entre voitures ou entre voitures et piétons.
4. **Simulation du temps réel :** Faire en sorte que la simulation utilise des intervalles de temps pour simuler le passage du temps de manière plus réaliste.

Conclusion

Cette simulation de ville permet de modéliser des voitures, des piétons, des bâtiments et des interactions simples comme le déplacement et l'arrêt aux bâtiments. Elle constitue une base solide pour des simulations urbaines plus complexes et montre comment utiliser la **POO (Programmation Orientée Objet)** pour créer des systèmes interactifs en Python.