Asie– 2024 – sujet1 - Correction

Exercice 1 (6 points)

---

1. Création des maisons m1 et m2

```
m1 = Maison(1)
m2 = Maison(3.5)
```

👉*Commentaire : On crée deux objets de type* `Maison` *en appelant le constructeur de la classe avec l'abscisse en paramètre.*

---

2. Création de l'antenne a

```
a = Antenne(2.5, 1)
```

👉*Commentaire : On suppose que le constructeur de* `Antenne` *prend en paramètres la position puis le rayon. L'antenne est placée en 2.5 avec un rayon d'action de 1.*

---

3. Représentation graphique (creation_rue)

Appel : `creation_rue([0, 2, 3, 4, 5, 7, 9, 10.5, 11.5])`

Maisons placées aux abscisses : 0 --- 2 - 3 - 4 - 5 --- 7 --- 9 - 10.5 - 11.5

👉*Commentaire : Les maisons sont simplement positionnées aux abscisses indiquées. La fonction trie d'abord la liste, donc elles seront bien ordonnées sur l'axe.*

---

4. Compléter la fonction creation_rue

```
def creation_rue(pos):
    pos.sort()
    maisons = []
    for p in pos:
        m = Maison(p)
        maisons.append(m)
    return maisons
```

👉*Commentaire :*

- `append(m)` *ajoute l'objet maison à la liste.*
- *On retourne la liste complète d'objets.*
- *Le tri garantit que les maisons sont classées par abscisse croissante.*

---

5. Méthode couvre

```
def couvre(self, maison):
    distance = abs(self.get_pos_antenne() - maison.get_pos_maison())
    return distance <= self.get_rayon()
```

👉*Commentaire :*

- *On calcule la distance entre l'antenne et la maison.*
- *On utilise les* getters *pour respecter l'encapsulation.*
- *L'antenne couvre la maison si la distance est inférieure ou égale au rayon.*

---

6. Résultat de strategie_1

Après exécution :

```
maisons = creation_rue([0, 2, 3, 4, 5, 7, 9, 10.5, 11.5])
antennes = strategie_1(maisons, 2)
print([a.get_pos_antenne() for a in antennes])
```

Résultat : [0, 3, 7, 10.5]

👉*Commentaire : La stratégie 1 place une antenne dès qu'une maison n'est pas couverte, généralement à la position de la maison courante.*
Cela conduit ici à 4 antennes.

---

7. Nouvelle stratégie (strategie_2)

Principe : Si une maison d'abscisse x n'est pas couverte, on place l'antenne en **x + r**.

Placement des antennes (rayon = 2)

Maisons : [0, 2, 3, 4, 5, 7, 9, 10.5, 11.5]

Étapes :

- Maison 0 → antenne en 2 (couvre jusqu'à 4)
- Maison 5 non couverte → antenne en 7 (couvre jusqu'à 9)

- Maison 10.5 non couverte → antenne en 12.5

Positions finales : `[2, 7, 12.5]`

👉*Commentaire : Cette stratégie est plus efficace : seulement 3 antennes au lieu de 4.*

---

8. Implémentation de strategie_2

```
def strategie_2(maisons, rayon):
    antennes = []
    for maison in maisons:
        couverte = False
        for antenne in antennes:
            if antenne.couvre(maison):
                couverte = True
                break
        if not couverte:
            position = maison.get_pos_maison() + rayon
            antennes.append(Antenne(position, rayon))
    return antennes
```

👉*Commentaire :*

- *On parcourt les maisons dans l'ordre.*
- *Si aucune antenne existante ne couvre la maison, on en place une nouvelle à* `x + rayon`.
- *Cette stratégie correspond à l'algorithme glouton optimal pour ce problème.*

---

9. Comparaison des coûts

Soit n le nombre de maisons.

Strategie_1 :

- Pour chaque maison, on peut tester plusieurs antennes.
- Pire cas : environ n antennes → complexité **O(n²)**.

👉*Commentaire : Double boucle potentielle (maisons × antennes).*

Strategie_2 :

- Chaque maison est parcourue une seule fois.
- Chaque antenne couvre un intervalle maximal.

Complexité : **O(n)**.

👉 **Commentaire :** On ne revient jamais en arrière. C’est un algorithme glouton optimal en temps linéaire

---