Introduction

Ce rapport individuel présente l'analyse de mes contributions au projet de SAÉ visant à développer une application web de gestion du matériel d’un laboratoire de paléontologie. L'objectif principal de l'outil est de rendre plus efficace pour gérer les fouilles et les échantillons.

Mes réalisations se sont concentrées sur l’implémentation des espèces hypothétiques et avérées (classes, calcul de distance), la reconstruction d’un arbre phylogénetique et le refactoring de tout les algorithmes python.

Compétences et Apprentissages Critiques Mobilisés

Les tableaux suivant détaille la manière dont mes actions ont satisfait les différents Apprentissages Critiques (AC) évalués lors de cette SAÉ. Vous pouvez retrouver mes conceptions pour la contribution au projet juste après les tableaux

1. RÉALISER (Conception et Implémentation)

AC	Contribution Réalisée	Justification
AC21.01	**Implémentation des classes EspeceHypothetique et EspeceAveree (getters, setters, etc.).** **Développement des algorithmes calculer_distance() et reconstruire_arbre()**	L'élaboration des classes et des algorithmes principaux a permis d'implémenter les spécifications fonctionnelles du projet (calcul de distance phylogénétique et construction d'arbre).
AC21.02	**Implémentation de la méthode __str__ s'appuyant sur _build_tree_string().**	L'application des principes d'ergonomie (pour le développeur) s'est traduite par une restitution visuelle claire de la structure de l'arbre sur le terminal, facilitant le débogage et la validation.

AC	Contribution Réalisée	Justification
AC21.03	**Utilisation de l'héritage (Avérée -> Hypothétique), des getters/setters, et de méthodes privées (_build_tree_string).** **Refactoring du code avec yapf, pylint et sonarlint.**	Adoption de bonnes pratiques de conception (programmation orientée objet) et de programmation (**R3.04 : Qualité de développement**) en utilisant des outils de *linting* pour assurer un code propre et maintenable.

2. OPTIMISER

AC	Contribution Réalisée	Justification
AC22.01	**Conception de la hiérarchie des classes (EspeceHypothetique avec une liste d'enfants).** **Développement de l'algorithme reconstruire_arbre() qui génère une structure arborescente.**	Le choix d'une structure de données arborescente, où les nœuds sont les espèces hypothétiques et les feuilles les espèces avérées, est directement adapté au problème de la phylogénie (**R3.02 : Développement efficace**).

4. GÉRER (Traitement des Données)

AC	Contribution Réalisée	Justification
AC24.03	**Implémentation des méthodes __str__ et _build_tree_string().**	L'organisation de la restitution de données complexes (l'arbre phylogénétique) a été réalisée via la programmation (méthode récursive _build_tree_string) pour permettre sa visualisation textuelle.

AC	Contribution Réalisée	Justification
AC24.04	**Développement de l'algorithme calculer_distance().**	L'algorithme a été conçu pour manipuler des données hétérogènes en gérant différents cas : la distance entre deux EspeceAveree (génome) et la distance impliquant au moins une EspeceHypothetique (enfants).

5. CONDUIRE (Gestion de Projet)

AC	Contribution Réalisée	Justification
AC25.02	**Rédaction du rapport commun (explication des fonctions et classes).**	La participation au rapport a permis de formaliser les besoins techniques et les solutions implémentées (mes classes et algorithmes) pour le reste de l'équipe et les évaluateurs.

6. COLLABORER (Travail d'Équipe)

AC	Contribution Réalisée	Justification
AC26.02	**Utilisation d'Outil de Gestion de projet : Google Drive, Github, ClickUp, Instagram**	Google Drive : Lister les tâches à faire, partager les ressources tel que les MCD, Diagramme UML, les rapports et des informations divers Github : Travailler en équipe sur un projet dans la programmation de l'application Instagram : Communiquer, Échanger des information entre les membres de l'équipe
AC26.03	**Rédaction du rapport commun.**	Mobilisation des compétences interpersonnelles pour échanger, expliquer mon travail et le consolider dans un document commun.
AC26.04	**Réalisation de ce rapport individuel**.	Ce document sert à rendre compte de mon activité professionnelle (mes tâches et les compétences mobilisées) de manière structurée et synthétique.

Conceptions Produites ou en Collaborations :

Implémentation de la classe EspeceHypothetique :

```
class EspeceHypothetique:
    """
    Espece Hypothétique
```

```
    Une espèce représentant un ancêtre commun à plusieurs espèces
    elle ne possède pas de génome
    """

    def __init__(self, nom_e: str, nom_sci: str):
        """
        Création de la classe Espece
        attributs:
        nom_e -> nom de l'espèce hypothétique
        nom_sci -> nom scientifique de l'espèce hypothétique
        enfants -> les espèces enfant de l'espèce hypothétique
        """
        self.nom_e = nom_e
        self.nom_sci = nom_sci
        self.enfants = []
```

Implémentation de la classe EspeceAveree :

```
class EspeceAveree(EspeceHypothetique):

    """
    Especе Avérée
    Une espèce représentable par un ou plusieurs animaux
    elle possède donc un génome
    """

    def __init__(self, nom_ea: str, nom_sci: str, genome: str):
        """
        Creation de la classe Espece_Averee
        attributs:
        genome -> ensemble du materiel genetique de l'espèce avérée
        """
        super().__init__(nom_ea, nom_sci)
        self.genome = genome

    # implémentation des getters et des setters

    def get_genome(self):
        return self.genome

    def set_genome(self, genome):
        self.genome = genome

    def add(self, enfant):
        self.enfants.append(enfant)

    def get_type_espece(self) -> str:
        return "avérée"
```

:

```
def calculer_distance(espece1, espece2):
    """
    Calcule la distance entre deux espèces, qu'elles soient avérées ou
hypothétiques.

    Param espece1 : Une espèce
    Param espece2 : Une autre espèce

    Return : la distance entre espece1 et espece2
    """
    #Dans le cas ou les 2 espèces sont les mêmes
    if espece1 is espece2:
        return 0
    #Dans le cas où les deux espèce sont des espèces avérées
    if isinstance(espece2, EspeceAveree) and isinstance(espece1, EspeceAveree):
        return mutation_replacement_distance(espece1.get_genome(),
                                             espece2.get_genome())

    #Dans le cas où espèce1 est hypothétique
    elif not isinstance(espece1, EspeceAveree):
        if not espece1.enfants:
            if not espece2.enfants:
                return 1.0
            return calculer_distance(espece1=espece2, espece2=espece1)

        somme_distances = 0
        for enfant in espece1.enfants:
            somme_distances += calculer_distance(enfant, espece2)
        return somme_distances / len(espece1.enfants)

    #Dans le cas où espèce2 est hypothétique (et espece1 est avérée)
    elif not isinstance(espece2, EspeceAveree):

        return calculer_distance(espece1=espece2, espece2=espece1)

    return 0
```

méthode reconstruire_arbre() :

```
def reconstruire_arbre(liste_especes_de_depart: list) -> EspeceHypothetique:
    """
    Construit un arbre phylogénétique à partir d'une liste d'espèces
    (qui sont les feuilles de l'arbre).
    """

    noeuds_actifs = liste_especes_de_depart.copy()
    while len(noeuds_actifs) >= 2:
        paire_min = trouver_paire_minimale(noeuds_actifs)
```

```
    noeud1, noeud2 = paire_min
    noeuds_actifs.remove(noeud1)
    noeuds_actifs.remove(noeud2)
    #On crée leur parent
    nom_parent = f"({noeud1.get_nom()} et {noeud2.get_nom()})"
    nom_sci_parent = f"({noeud1.get_nom_sci()}, {noeud2.get_nom_sci()})"

    nouvel_ancetre = EspeceHypothetique(nom_parent, nom_sci_parent)
    nouvel_ancetre.add_enfants(noeud1)
    nouvel_ancetre.add_enfants(noeud2)

    noeuds_actifs.append(nouvel_ancetre)

  #On retourne la racine de l'arbre
  return noeuds_actifs[0]
```

méthode trouver_paire_minimale() :

```
def trouver_paire_minimale(noeuds: list) -> tuple:
  """
  Trouve la paire d'espèces avec la distance minimale entre elles.
  """
  distance_min = None
  paire_min = None
  # for imbriqué afin de vérifier la distance sur tout les noeuds
  for noeud1 in noeuds:
    for noeud2 in noeuds:
      # tant que les 2 noeuds sont différents chacun de l'autre
      if noeud1 != noeud2:
        distance_init = calculer_distance(noeud1, noeud2)
        if distance_min is None or distance_min > distance_init:
          distance_min = distance_init
          paire_min = (noeud1, noeud2)
  return paire_min
```

Conclusion

Ce projet a permis de développer les **composants algorithmiques et structurels fondamentaux** pour la modélisation phylogénétique. L'implémentation a été centrée sur deux axes majeurs :

1. **La Conception Orientée Objet (AC21.03) :** La création des classes EspeceHypothetique et EspeceAveree avec héritage a fourni une base de données flexible et adaptée au problème (AC22.01).
2. **Le développement efficace (R3.02) :** La conception d'algorithmes récursifs complexes, notamment calculer_distance() et reconstruire_arbre(), a permis de résoudre le problème central du calcul de distance et de la génération automatique d'arbres.

La gestion de différents types d'entrées (avérée vs. hypothétique) dans la méthode calculer_distance() a démontré une manipulation adéquate de données hétérogènes (AC24.04).

De plus, un effort a été consacré à la **qualité du développement (R3.04)** par le refactoring systématique (pylint, yapf) et à la lisibilité des résultats via la visualisation textuelle de l'arbre (AC24.03). Enfin, le travail a été mené en collaboration, en intégrant des composants externes (AC26.02) et en participant activement à la documentation collective (AC26.03, AC26.04).