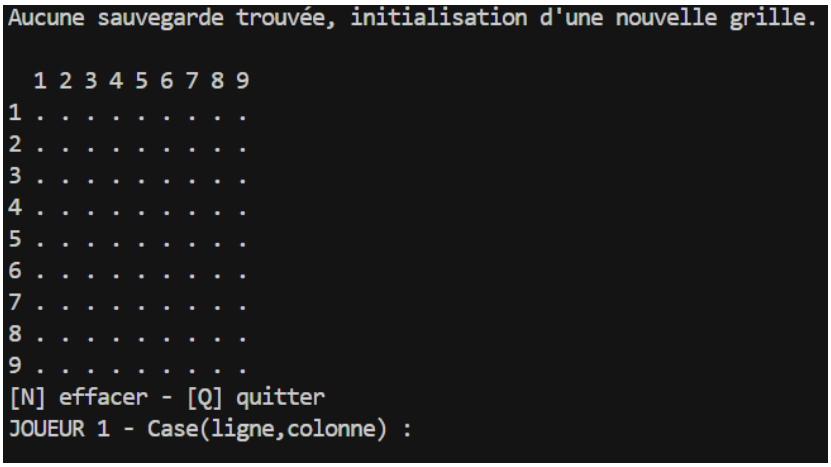
Analyser un problème avec méthode

```python
def est_fin_de_partie_methode1(p_lgn, p_col):
    # Implémentez la logique pour vérifier si un joueur a gagné
    # Vérifie les lignes
      Ctrl+L to chat, Ctrl+K to generate
    for lig in range(9):
        ligne = "".join(grille[lig])
        if "ooooo" in ligne or "xxxxx" in ligne: # Vérifie si la ligne contient cinq 'o' ou cinq 'x' de manière consécutif
            return True  # Un joueur a gagné
    # Vérifie les colonnes
    for col in range(9):
        colonne = "".join(grille[lig][col] for lig in range(9)) # Pour chaque colonne, on crée une chaîne de caractères en concaténant les éléments de la colonne
        if "ooooo" in colonne or "xxxxx" in colonne: # Vérifie si la colonne contient cinq 'o' ou cinq 'x' de manière consécutif
            return True  # Un joueur a gagné

    # Vérifie les diagonales  (de haut en bas, gauche à droite)
    for start in range(-4, 5):  # Décalage pour avoir les diagonales qui ont au minimu 5 valeur
        diagonale = []
        for i in range(9):
            j = i + start
            if 0 <= i < 9 and 0 <= j < 9: # Permet de vérifier si nous respectons les limites de la grille
                diagonale.append(grille[i][j])
        if "ooooo" in "".join(diagonale) or "xxxxx" in "".join(diagonale):  # Vérifie si la diagonale contient cinq 'o' ou cinq 'x' de manière consécutif
            return True # Un joueur a gagné

    # Vérifie les diagonales inversé (de haut en bas, droite à gauche)
    for start in range(-4, 5):  # Décalage pour avoir les diagonales qui ont au minimu 5 valeur
        diagonale = []
        for i in range(9):
            j = start - i
            if 0 <= i < 9 and 0 <= j < 9:
                diagonale.append(grille[i][j])
        if "ooooo" in "".join(diagonale) or "xxxxx" in "".join(diagonale): # Vérifie si la diagonale inversée contient cinq 'o' ou cinq 'x' de manière consécutif
            return True # Un joueur a gagné
```

```python
def est_fin_de_partie_methode2(p_lgn, p_col):

    # Implémentez une autre méthode pour vérifier si un joueur a gagné
    def compter_alignes(ligne, colonne, delta_ligne, delta_colonne, symbole):
        # Compte les symboles alignés dans une direction donnée
        count = 0
        i, j = ligne, colonne
        while 0 <= i < 9 and 0 <= j < 9 and grille[i][j] == symbole:
            count += 1  # Trouvé un symbole identique
            i += delta_ligne  # avance dans la direction spécifiée
            j += delta_colonne
        return count # Retourne le nombre de symboles alignés

    symbole = grille[p_lgn][p_col]  # Récupérer le symbole joué
    if symbole not in ["o", "x"]:
        return False  # Pas de pion joué ici

    # Vérification horizontale
    total = compter_alignes(p_lgn, p_col, 0, 1, symbole) + compter_alignes(p_lgn, p_col - 1, 0, -1, symbole) # Permet de compter les symboles identiques
    #alignés à droites et à gauche à partir de sa position actuelle
    if total >= 5: # Si le total est supérieur ou égal à 5 la parite est terminée
        return True

    # Vérification verticale
    total = compter_alignes(p_lgn, p_col, 1, 0, symbole) + compter_alignes(p_lgn - 1, p_col, -1, 0, symbole) # Permet de compter les symboles identiques
    #alignés en haut et en bas à partir de sa position actuelle
    if total >= 5: # Si le total est supérieur ou égal à 5 la parite est terminée
        return True
```

```python
# Vérification diagonale  (de haut en bas, gauche à droite)
total = compter_alignes(p_lgn, p_col, 1, 1, symbole) + compter_alignes(p_lgn - 1, p_col - 1, -1, -1, symbole)
if total >= 5: # Si le total est supérieur ou égal à 5 la parite est terminée
    return True

# Vérification diagonale inversée (de haut en bas, droite à gauche)
total = compter_alignes(p_lgn, p_col, 1, -1, symbole) + compter_alignes(p_lgn - 1, p_col + 1, -1, 1, symbole)
if total >= 5:# Si le total est supérieur ou égal à 5 la parite est terminée
    return True
return False
```



Je suis capable d'analyser un problème de manière structurée et de le traduire en éléments algorithmiques simples. Dans la SAÉ 1.02 « Comparaison d'approches algorithmiques » moi et mes camarades nous avons conçu deux fonctions pour retrouver le/la vainqueur (e) du Morpion. Pour ce faire, nous avons décomposé le problème en différents axe puis, nous nous sommes appuyés sur nos structures de données une grille 2D pour parcourir celle-ci et retrouver facilement les différents alignements.

Comparer des algorithmes pour des problèmes classiques

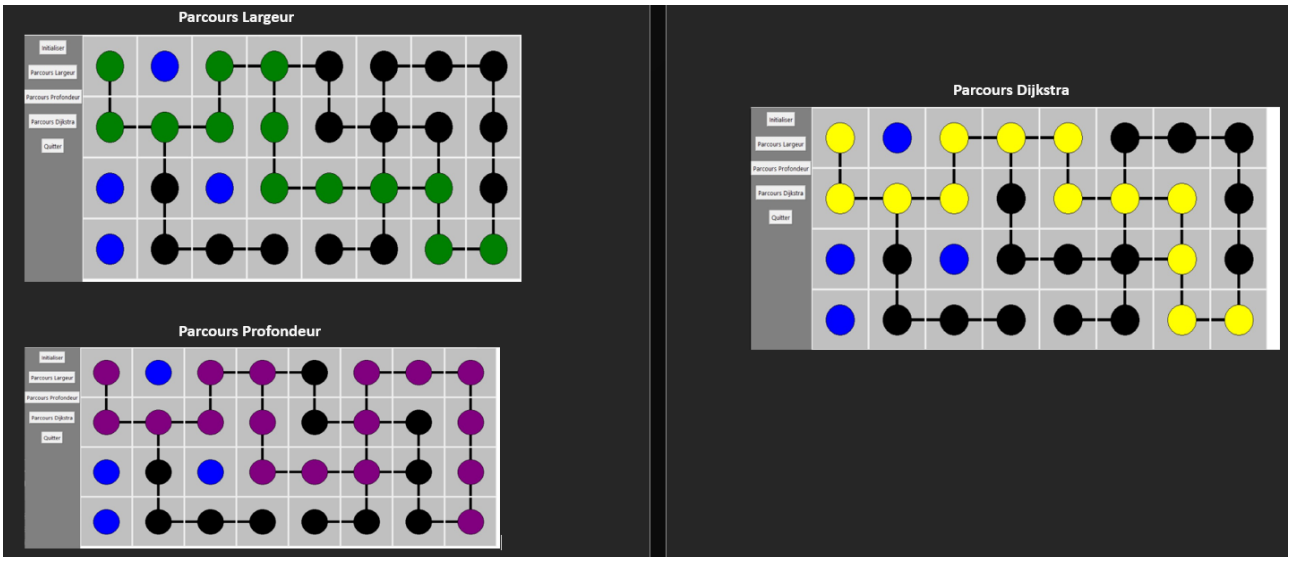
Tâches à réaliser :

1. Modéliser le plateau comme une grille d'objets noeuds en mémoire

2. Initialiser les liaisons sous la forme d'un arbre quaternaire d'objets (Noeud) Python, reliés les uns aux autres.
   Chaque Noeud aura pour propriétés :
   - nom : string — # *coordonnées (colonne, ligne)*
   - poids : entier — # *uniquement pour Dijkstra*
   - nord, sud, est, ouest : Noeud — # *noeuds adjacents*
   - statut : string — # *"aucun" / "visité" / "actif"*

3. Implémenter la recherche de la sortie avec chacun des algorithmes suivants :
   - parcours en profondeur
   - parcours en largeur
   - algorithme de Dijkstra

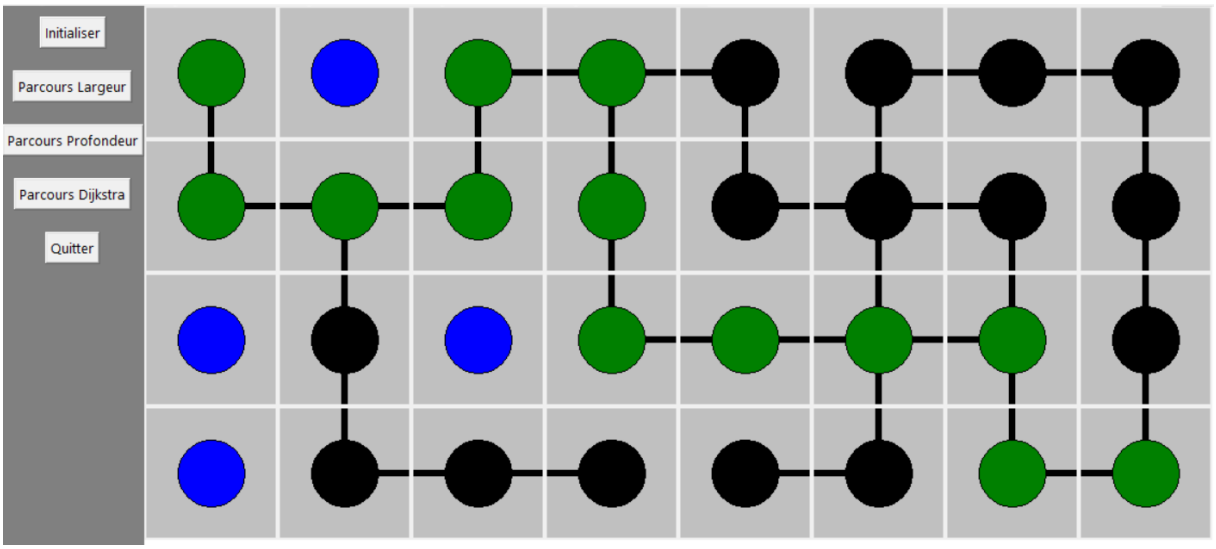
   Le parcours du graphe sera journalisé dans un fichier.



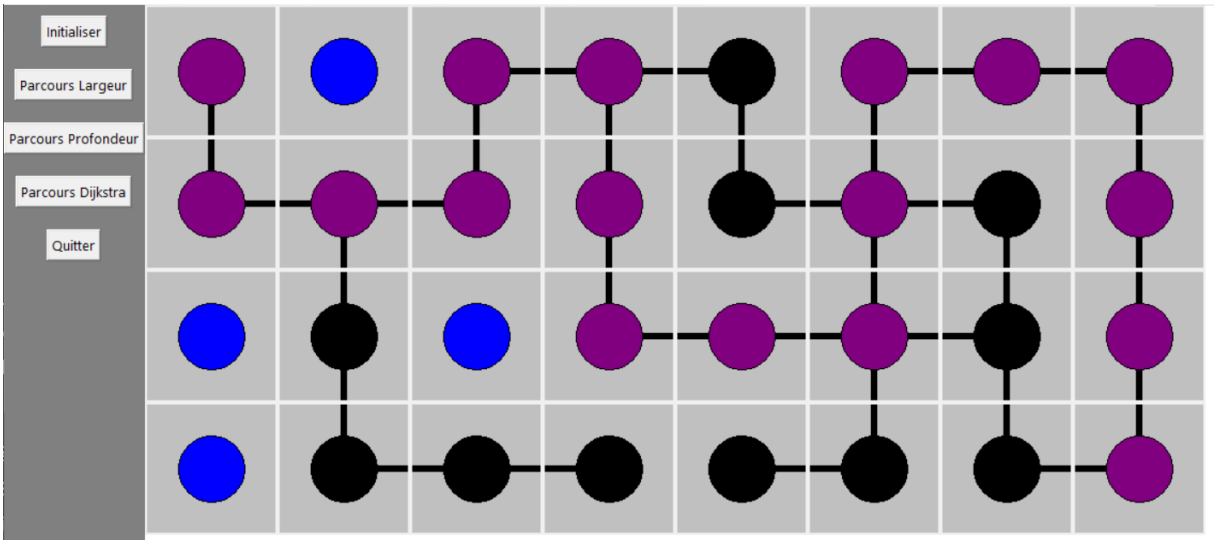
Je suis en capacité de comparer des algorithmes pour des problème classiques. Dans la SAÉ 2.02 « Exploitation algorithmique d'un problème » j'ai analysé les performances du parcours en profondeur, en largeur et de l'algorithme de Dijkstra pour résoudre un labyrinthe, en mettant en évidence leurs avantages et limites selon le contexte d'utilisation.

Formaliser et mettre en œuvre des outils mathématiques pour l'informatique

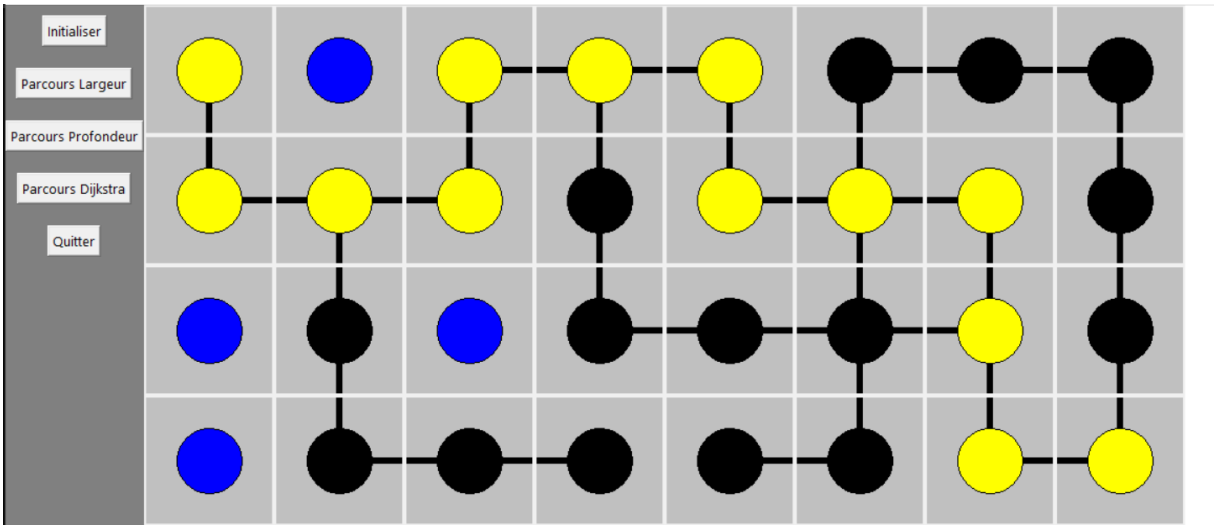
Parcours Largeur



Parcours Profondeur



Parcours Dijkstra



Dans le cadre d'un projet de résolution de labyrinthe, nous avons modélisé un plateau sous forme de graphe, puis implémenté et comparé trois algorithmes (Parcours Largeur, Parcours Profondeur et Dijkstra) pour explorer ce graphe. Ce travail m'a permis d'appliquer concrètement des concepts de mathématique discrètes, de structures de données, et d'analyse algorithmique.