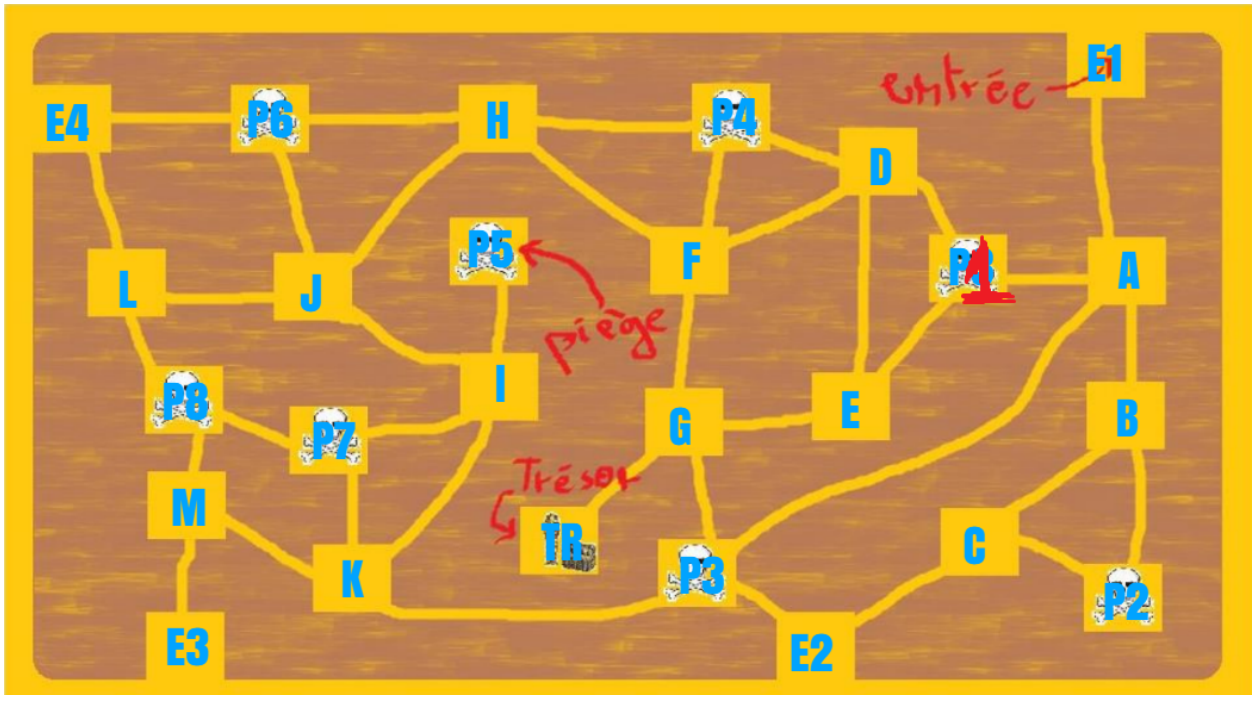
Projet AI (labyrinth)

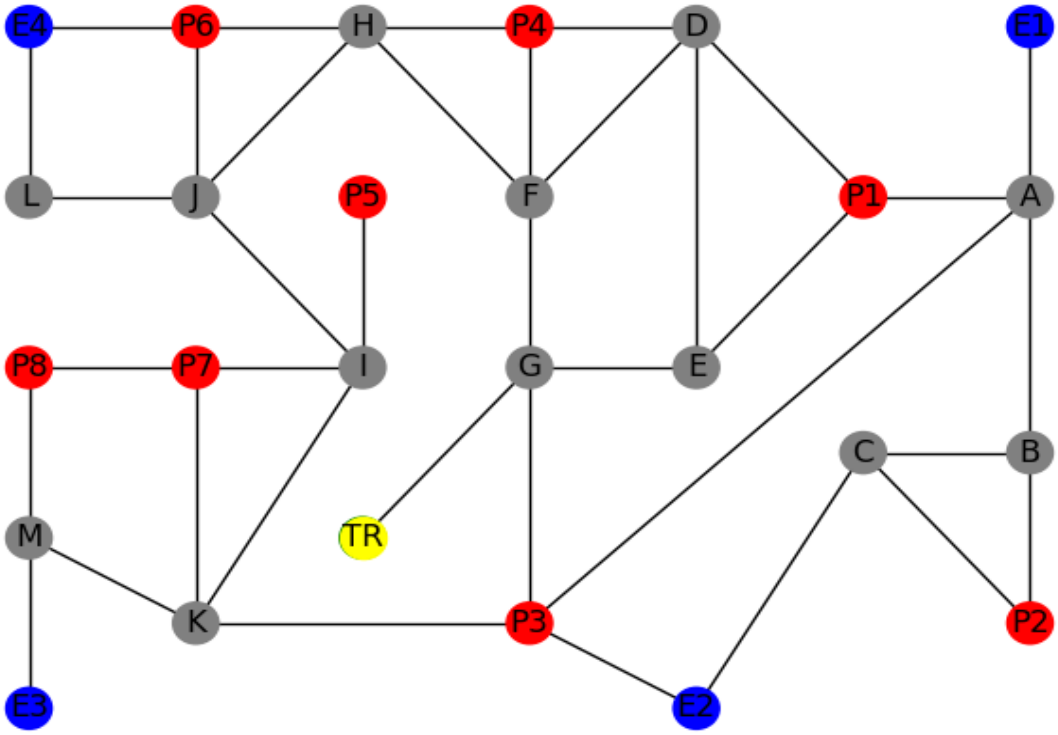
I-Intoduction:

Le but de ce projet est de résoudre un labyrinthe à l'aide de l'algorithme de recherche en profondeur. Cette méthode consiste à explorer le labyrinthe en profondeur en suivant un chemin jusqu'à ce qu'il ne soit plus possible de continuer, puis en revenant en arrière jusqu'à trouver un nouveau chemin à explorer. L'algorithme de recherche en profondeur est une méthode simple et efficace pour résoudre un labyrinthe, mais peut parfois entraîner une exploration exhaustive de toutes les possibilités avant de trouver la sortie.

Les noms des noeuds:



presentation en graph:



II-Methologies

1. Choix de l'algorithme de recherche

Nous avons opté pour l'algorithme de recherche en profondeur pour résoudre le labyrinthe en partant de la trésorerie et en remontant vers les entrées. Cette méthode nous a semblé la plus appropriée pour explorer le labyrinthdo e de manière systématique et efficace, tout en garantissant la découverte de toutes les solutions possibles.

2. Implémentation de l'algorithme de recherche

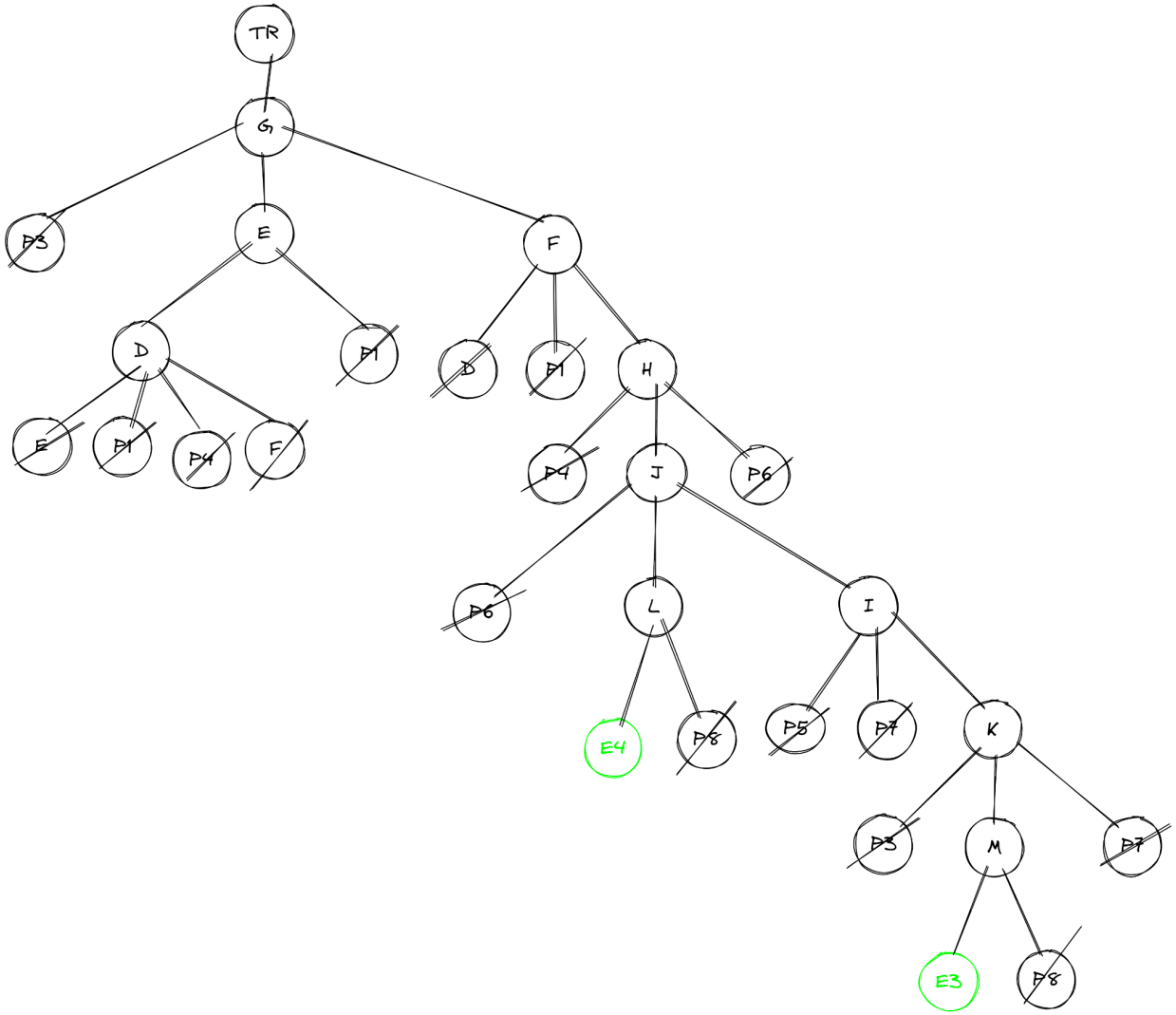
```python

# Algorithme recherche profendeur
def rp(graph, noeud_initial, noeuds_finals, pieges=None):
    ferme = set()  # les noeuds visités
    ouvert = [(noeud_initial, [noeud_initial])]  # les noeuds a visiter
    solutions = []  # liste des solutions
    nx.draw(G, pos=pos, with_labels=True, node_color=list(node_colors.values()))

    while ouvert:
        (noeud, chemin) = ouvert.pop()  # retirer la dernier noeud visité

        # Front-end: Mise a jour des chemins en cours d'exploitation
        nx.draw_networkx_edges(G, pos=pos, edgelist=chemin_vers_liens(chemin), edge_color="yellow", width=3.0)
        nx.draw_networkx_nodes(G, pos=pos, nodelist=chemin, node_color="yellow",)
        plt.pause(VITTESSE)

        if noeud not in ferme:  # vérifier si la noeud est visité
            ferme.add(noeud)  # ajouter la noeud avec les noeud visité
            if noeud in noeuds_finals:  # verifier si la noeud est une noeud final
                solutions.append(chemin)  # ajouter le chemin a la liste des solutions

            else:
                for voisin in graph[noeud]:  # explorer le noeud voisins
                    if voisin not in ferme and (not pieges or voisin not in pieges):
                        # verifier si la noeud est visité ou c'est une noeud piége
                        ouvert.append((voisin, chemin + [voisin]))  # ajouter la noeud voisin avec le nouvaux chemin

    colors = ['#1e4620', '#276749',  '#34a165',  '#7fdbb3', ] #différents degré de vert pour chacque solution


    for i, s in enumerate(solutions):
        # Front-end: Mise a jour des chemins solutions
        nx.draw_networkx_edges(G, pos=pos, edgelist=chemin_vers_liens(s), edge_color=colors[i], width=20.0/(i+1))
        nx.draw_networkx_nodes(G, pos=pos, nodelist=s, node_color="green",)
        plt.pause(VITTESSE)


    return solutions  # return de tous les solution
```

III-Résultat

Nous avons testé notre méthode de résolution du labyrinthe en partant de la trésorerie et en remontant vers les entrées est on a eu ces deux chemin comme solution:

- **E4 L J H F G TR**
- **E3 M K I J H F G TR**

résultat théorique:



visualisation du l'algorithme: