Rapport du Travail Collaboratif en IA:
Implémentation d'algorithme de Recherche Heuristique A*
En JAVA & réalisation de son interface graphique.

Realisé par:

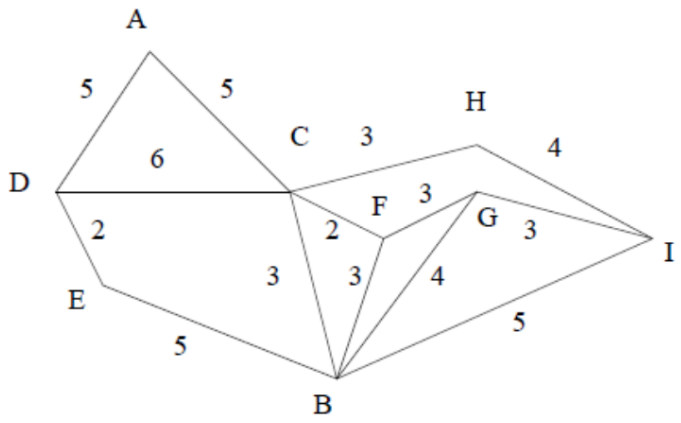
AGLMOUS Achraf	**#3**	**2ACI-A**
BENAZZOU Adnane	**#10**	
KASSI Khalid	**#31**	
LACHHEB Hajar	**#32**	
REGUIG Jamila	**#44**	

Professeur:
Mme. Najima DAOUDI
2020/2021

Plan du Rapport:

Enoncé du Problème:

Implémentation d'un algorithme de recherche heuristique en java & application sur l'exercice suivant:



En considérant la carte suivante. Le but est de trouver le chemin le plus court de A vers I. Le coût de chaque connexion est indiqué. Deux heuristiques h1 et h2 sont données comme suit:

Noeud	A	B	C	D	E	F	G	H	I
h_1	10	5	5	10	10	3	3	3	0
h_2	10	2	8	11	9	6	3	4	0

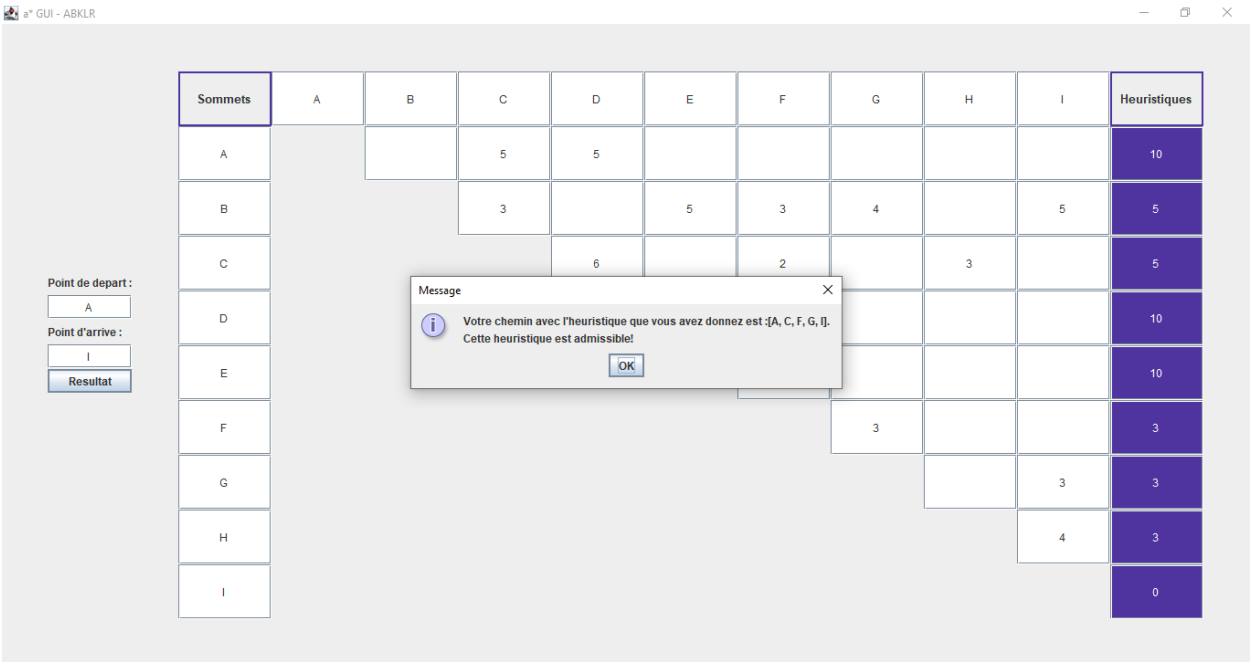
Problématique:

1. Est-ce que h1 et h2 sont admissibles? Justifiez.
2. Est-ce que h3 = max (h1, h2) est admissible?
3. Appliquez la recherche A* en utilisant h1. Donnez la suite des nœuds développés.
4. Appliquez la recherche A* en utilisant h2. Donnez la suite des nœuds développés.
5. Appliquez la recherche A* en utilisant h3. Donnez la suite des nœuds développés.

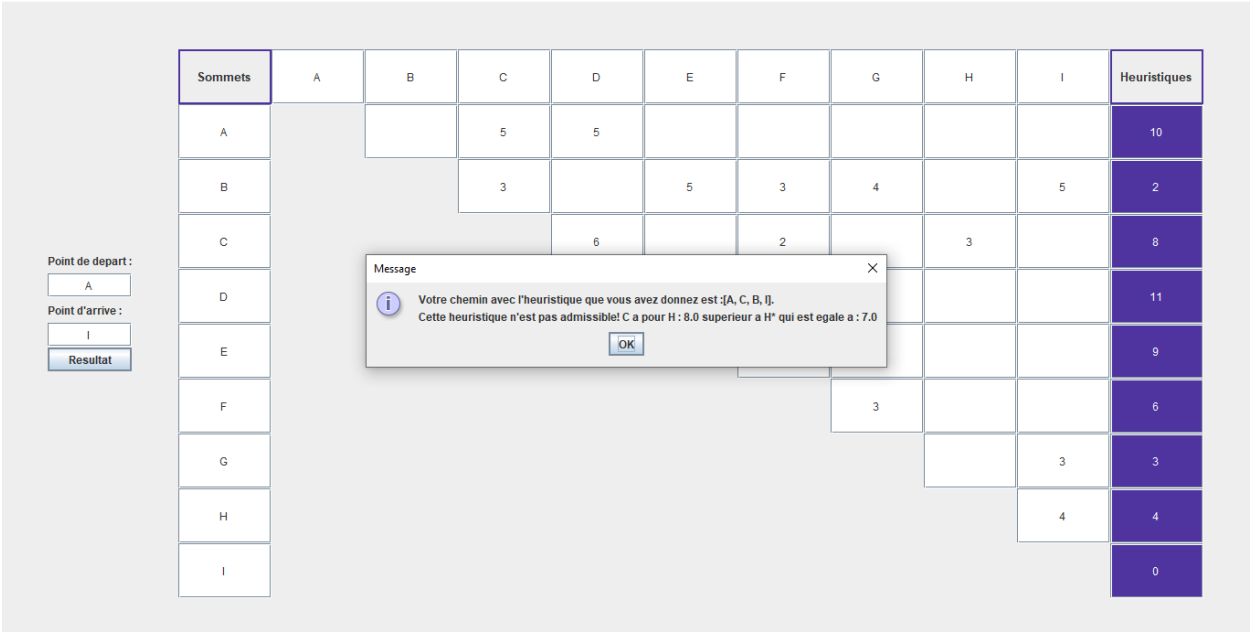
Solution:

Dans l'exercice on a **9 sommets** (A à I), on insère ce nombre dans la zone dédiée dans la fenêtre qui s'affiche lors de l'exécution du programme:

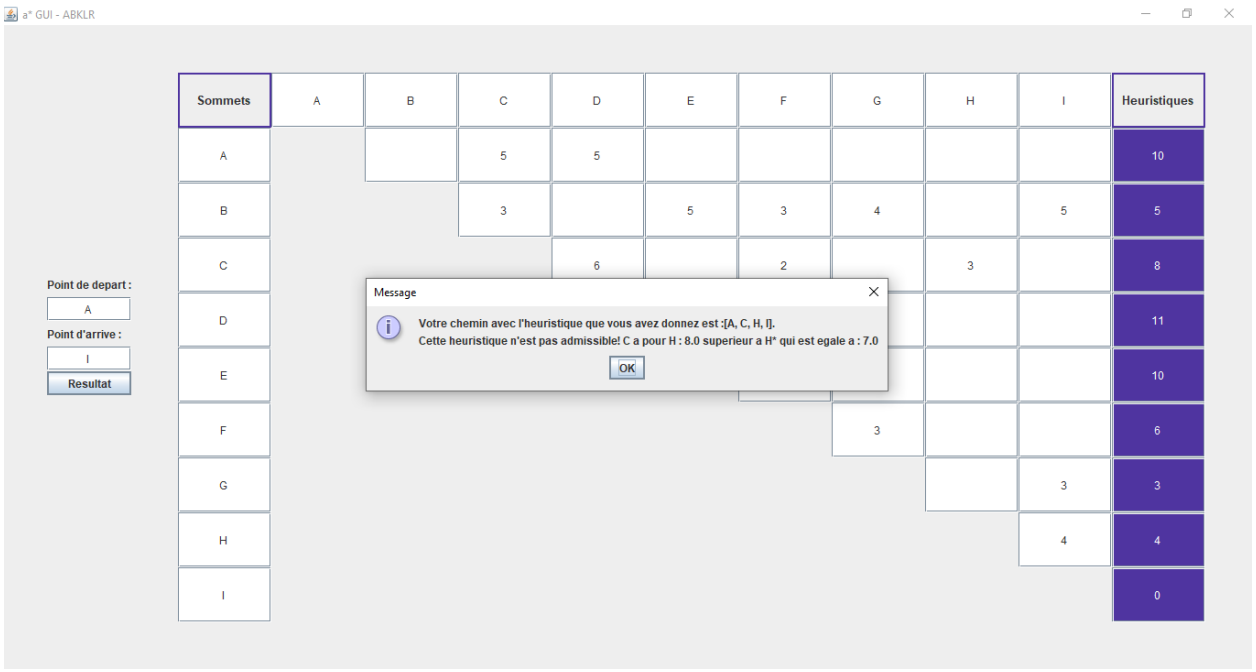
Après on passe à l'étape suivante, qui consiste à donner les noms des sommets du graphes ainsi que les poids qui caractérisent les arcs du graphe et la fonction heuristique à évaluer.(ie: h1, h2 et h3 dans notre exercice).
On doit ainsi renseigner les sommets de départ et d'arrivée dont on veut trouver la suite des nœuds développés par notre heuristique.



Sommets	A	B	C	D	E	F	G	H	I	Heuristiques
A			5	5						10
B			3		5	3	4		5	5
C				6		2		3		5
D										10
E										10
F							3			3
G									3	3
H									4	3
I										0



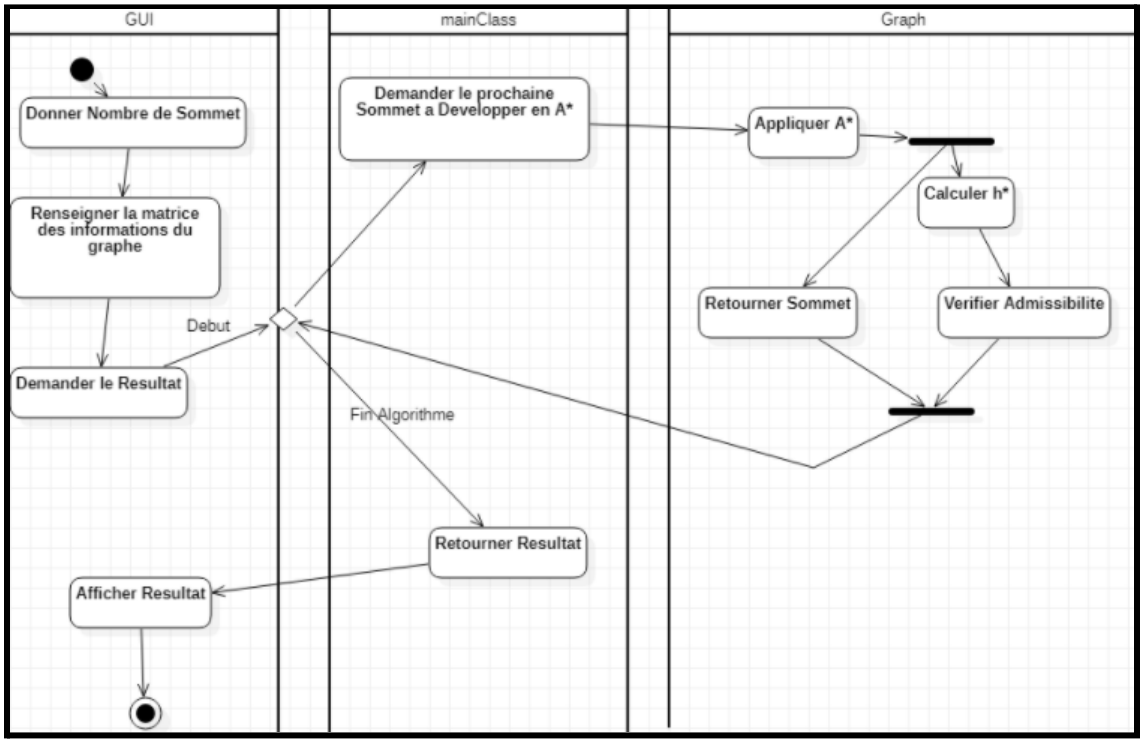
Sommets	A	B	C	D	E	F	G	H	I	Heuristiques
A			5	5						10
B			3		5	3	4		5	2
C				6		2		3		8
D										11
E										9
F							3			6
G									3	3
H									4	4
I										0



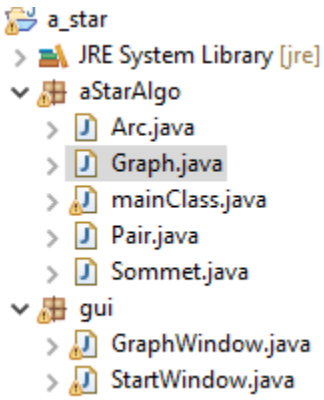
Comme le montre les captures d'écrans, le programme va donner la suite développées suivant notre fonction heuristique en appliquant l'algorithme a star ainsi que l'admissibilité ou non de cette heuristique avec sa justification si elle n'est pas admissible.

Processus:

I - Modélisation UML du fonctionnement du programme



II - La structure du programme (Classes & interfaces):



- **Deux packages:**

- Package **"aStarAlgo":** qui contient le code de l'implémentation d'algorithme A*:

    - **Classe Sommet:**
        - Cette classe consiste à modéliser les sommets de notre graphes en stockant le nom du sommet ainsi que les divers fonctions au niveau de chaque sommet(g le coût jusqu'à présent, h le coût estimé, h* le vrai coût et f qui est le coût total estimé qu'utilisera notre algorithme).
    - **Classe Arc:**
        - Cette classe consiste en un type qu'on va utiliser pour stocker les arcs qui relient le graphes en ArrayList, elle va contenir les 2 sommets ainsi que le poids de l'arc qui relie ces sommets.
    - **Classe Graph:**
        - C'est la classe qui modélise notre graphe et qui va contenir l'ensemble des méthodes qu'on peut appliquer sur le graphe a savoir: nos sommets, arc, liste d'adjacences ([Sommet 1[Adjacents], Sommet 2[Adjacents],...]), notre implémentation de l'algorithme qui trouvera notre prochain sommet ainsi que notre implémentation de Dijkstra afin de trouver h* pour chaque sommet.
    - **Classe Pair:**
        - C'est une classe qu'on a créé pour Dijkstra principalement car on utilisera une PriorityQueue qui ordonnera nos sommets selon la distance donnée donc on aura besoin d'une classe comparable qui trie ces sommets en fonction de la distance.
        - On l'a utilisée aussi pour stocker notre liste d'adjacence sous forme Sommet 1[Pair(Adjacent 1, Poids), Pair(Adjacent 2, Poids)...]

- **Classe mainClass:**
  - Cette Classe sert principalement à récupérer l'input du GUI, Instancier notre Graphe et Recuperer la suite des sommets développés par notre algorithme A*, en effet c'est une boucle while qui se suffit à procéder suivant les sommets développés jusqu'à présent ainsi que leur adjacents. Elle retourne comme résultat une liste composée de 2 chaînes: la 1ere est le chemin développé et la 2ème est l'admissibilité de l'heuristique.

- Package **"gui"**: qui contient le code de l'interface graphique.
  - **StartWindow:**
    - La JFrame qui représente la première fenêtre de l'interface graphique.
      L'utilisateur est censé introduire le nombre des sommets dans son graphe + Une petite documentation explicative de la méthode d'utilisation de l'interface graphique.

  - **GraphWindow:**
    - La JFrame représente la deuxième fenêtre de l'interface graphique. L'utilisateur est censé introduire les noms des sommets, les poids des arcs, les heuristiques des sommets, et finalement le sommet de départ et d'arrivée.

Conclusion:

Est-ce que h1 et h2 sont admissibles? Justifiez.	-H1 admissible ($\forall$ X), H1(X)<=H1*(X). -H2 non admissible H2(C) > H2*(C).
Est-ce que h3 = max (h1, h2) est admissible?	-H3 non admissible H3(C) > H3*(C).
Appliquez la recherche A* en utilisant h1. Donnez la suite des nœuds développés.	ACFGI.
Appliquez la recherche A* en utilisant h2. Donnez la suite des nœuds développés.	ACBI.
Appliquez la recherche A* en utilisant h3. Donnez la suite des nœuds développés.	ACHI.