**Analyse**

***Trouver la ou les routes qui rapportent le plus de points entre des points données à parcourir dans une grille 2D.***

***Licence RGI - Groupe ERP CISCO :***

**Maël RHUIN**

**Sommaire**

**Approche**

***Sujet*** : Trouver la ou les routes qui rapportent le plus de points entre des points données à parcourir dans une grille 2D.

***Réflexion autour de l’algorithme de Dijkstra :***

L’[algorithme de Dijkstra](https://fr.wikipedia.org/wiki/Algorithme_de_Dijkstra) est une méthode qui permet de trouver le plus court chemin entre deux sommets d’un graphe pondéré par des réels positifs. Il a été inventé par le mathématicien et informaticien néerlandais [Edsger Dijkstra](https://fr.wikipedia.org/wiki/Edsger_Dijkstra) en 1956. L’idée principale de l’[algorithme](https://www.maths-cours.fr/methode/algorithme-de-dijkstra-etape-par-etape) est de maintenir un ensemble de sommets dont les distances minimales à la source sont connues, et d’ajouter progressivement le sommet le plus proche de la source à cet ensemble. Ci-après l’algorithme de Dijkstra en pseudo code :

// Entrée : un graphe G = (V, E) pondéré par des réels positifs et un sommet source s

// Sortie : un tableau dist qui contient les distances minimales de s à tous les autres sommets

// Initialisation

Pour chaque sommet v de V

dist[v] = +infini // Distance infinie à l'origine

visité[v] = faux // Sommet non visité

Fin pour

dist[s] = 0 // Distance nulle à la source

// Boucle principale

Tant qu'il existe un sommet non visité

u = le sommet non visité ayant la plus petite distance // Choix glouton

visité[u] = vrai // Marquer u comme visité

Pour chaque voisin v de u

si dist[u] + poids(u, v) < dist[v] // Relâchement des arêtes

dist[v] = dist[u] + poids(u, v) // Mise à jour de la distance de v

Fin si

Fin pour

Fin tant que

Retourner dist

**Cahier des charges**

La réalisation de cette analyse donnant lieu à un programme informatique est conditionnée par un cahier des charges :

Sur un plateau donné de largeur 20 et longueur 20 disponible dans le fichier Excel joint.

* Case négative (**noire**) : infranchissable
* Case positive : possible de se déplacer avec le coût indiqué
* Case stratégiques (**rouge**) :
* Case d'intérêts (**vert**) :

Construire un ***dataset*** contenant les routes (les plus courtes) et le coût de déplacement (sommes des cases traversées) entres :

* tous les points stratégiques de la carte
* tous les points d'intérêts de la carte
* tous les points stratégiques et les points d'intérêts

La personne arrive en coordonnées {x ; y}, x = 11 et y = 19, et doit se rendre aux points stratégiques 1, 3, 6 & 7 :

* Chaque point stratégique lui rapporte 30 points.
* Chaque point d’intérêt lui rapporte sa valeur donnée.

Déplacements autorisés : horizontaux et verticaux (pas de diagonales)

Calculer le ou les chemins pour que le personnage se rende aux lieux stratégiques indiqués et récolte au passage le maximum de points en passant par des lieux d'intérêts

Calcul des points

Points des lieux stratégiques + points des lieux d'intérêts traversés - poids de chaque déplacement

L'objectif est de trouver la ou les routes donnant le plus de point !

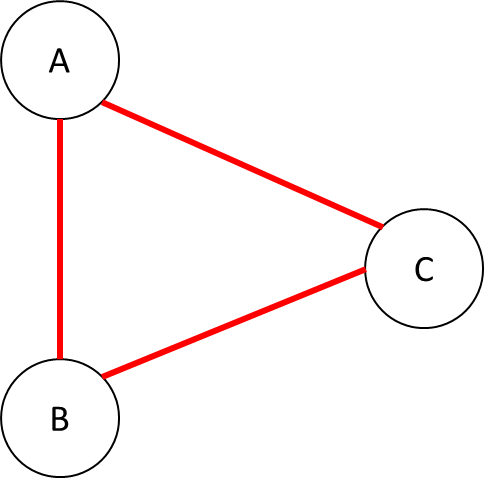
**Analyse de Niveau 0**

***Sujet*** : Trouver la ou les routes qui rapportent le plus de points entre des points données à parcourir dans une grille 2D.

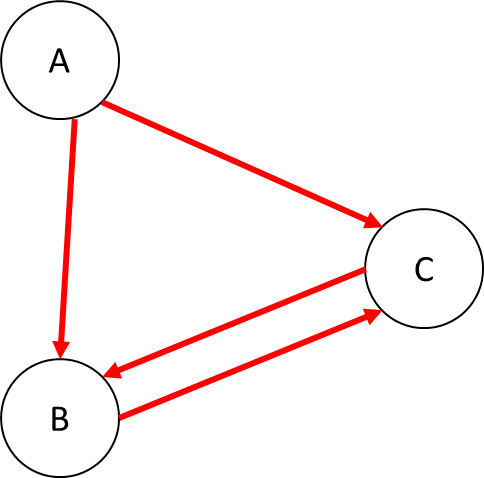
Après avoir trouvé une approche à la résolution de notre sujet. Il nous faut l’appliquer. Pour résumé l’algorithme de Dijkstra requiert un graphe orienté et pondéré (chaque arc possède un poids qui doit être ≥ 0.

***Qu'est-ce qu'un graphe ?***

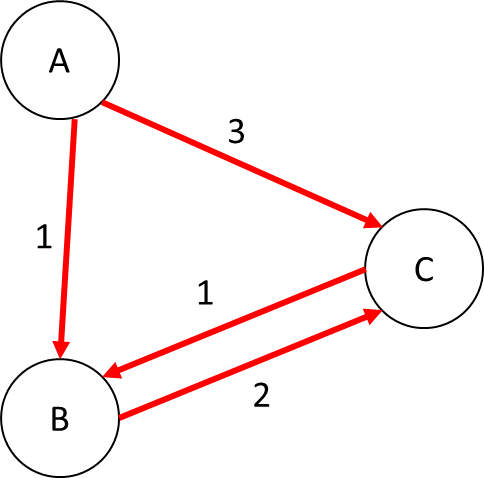
Un [graphe](https://fr.wikipedia.org/wiki/Graphe_(math%C3%A9matiques_discr%C3%A8tes)) en mathématiques est une structure composée d’objets appelés sommets et de relations entre eux appelées arêtes (ou arcs).

À gauche un graphe composé de trois sommets A, B, C relié par des arcs en rouge.

Un graphe ***orienté*** est un graphe où les arcs ont une direction représentée par une flèche.

 À gauche un graphe composé de trois sommets A, B, C relié par des arcs orientés en rouge.

Un graphe ***pondéré*** est un graphe où chaque arc porte un nombre appelé **poids**. Les poids peuvent représenter des coûts, des longueurs ou des capacités selon le problème.

 À gauche un graphe composé de trois sommets A, B, C relié par des arcs orientés et pondérés en rouge.

Il faut donc d’abord transformer notre matrice d’entiers en un graphe orienté et pondéré pour pouvoir lui appliquer Dijkstra.

***Comment faire ?***

Il nous faut découper le problème en plusieurs outils de résolution. Pour former notre graphe il nous faut des sommets, des arcs et des poids. Pour cela on décompose en objets :

Notre ***carte*** possède une retraduction en un graphe :

Un ***graphe*** est composé de sommets :

Un ***sommet*** possède un nom unique et des arcs orientés :

Un ***arc*** possède un sens (d’un sommet A vers B) et un poids positif.

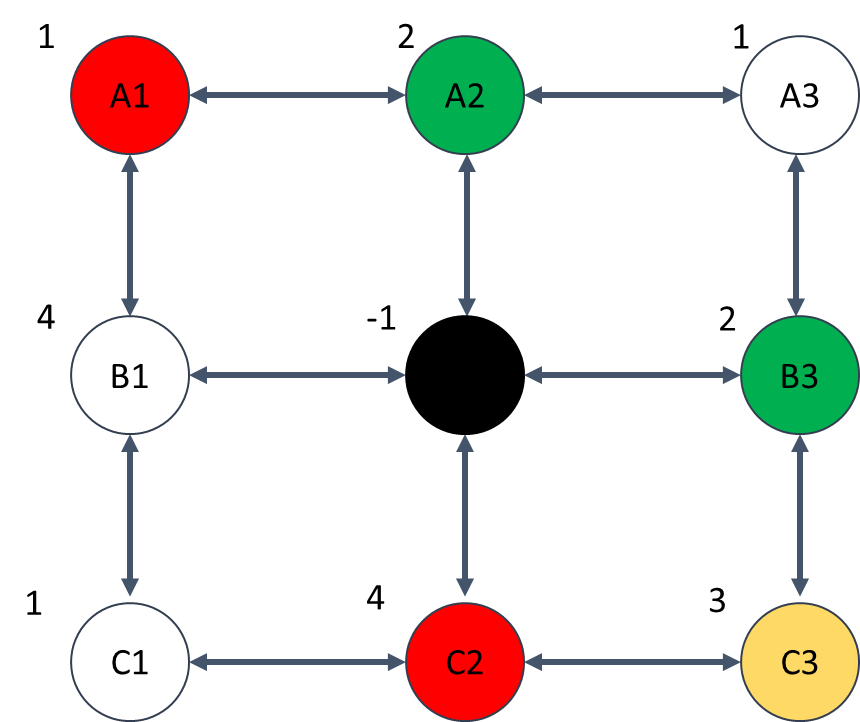
Soit une matrice d’entiers compris entre un minimum ***a*** et un maximum ***b*** de taille donnée par deux entiers ***l*** (largeur)et ***h*** (hauteur) :

***Exemple :***

***a = 1 et b = 4 ; l = 3 et h = 3***

On affecte à chaque ***case*** de cette matrice un nouveau ***sommet*** qui peut être un obstacle ou non, de manière à obtenir une liste de sommets :

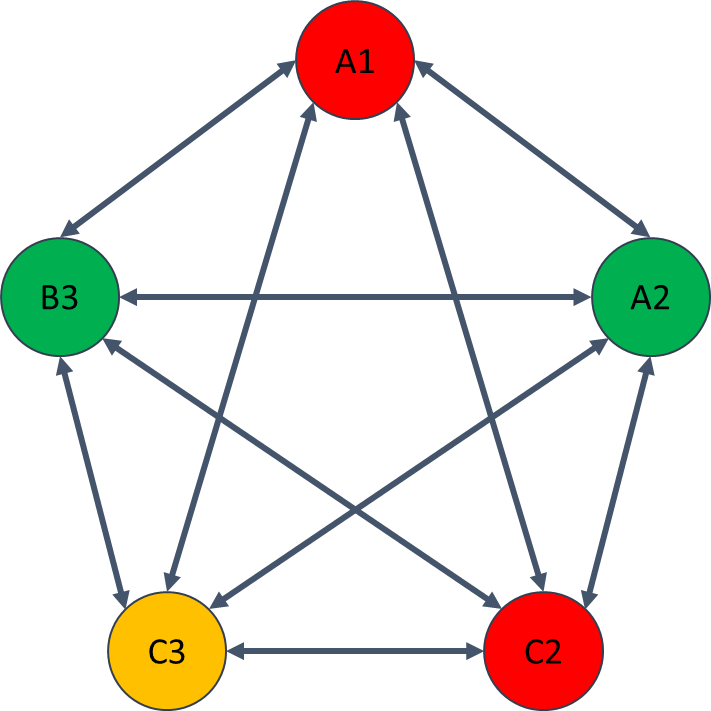
Pour chaque ***voisin*** (haut, bas, gauche, droite) de chaque ***sommet***, on crée un ***arc*** de ***poids*** correspondant à la ***case*** de notre matrice pour obtenir le graphe correspondant :



Sur ce graphe on y a placé pour exemple les points « types » du cahier des charges :

* En noir l’obstacle
* En rouge les points stratégiques
* En vert les points d’intérêts
* En jaune le point de départ

Comme Dijkstra requiert des arcs de poids ≥ 0, dans le cas des obstacles nous déclarons le sommet comme étant un obstacle et son poids entrant est définit à un nombre très grand proche de « l’infini ».

**** La suite de notre raisonnement va être de « minimisé ce graphe » en un plus petit ne contenant que les points qui nous intéresse. Chaque arc sera cette fois-ci de poids correspondant au score calculé du chemin le plus court entre 2 sommets. Chaque sommet de ce graphe est connecté à un autre. On appelle ce type de graphe : « graphe complet » .

On peut ensuite à partir de ce graphe sélectionner en le parcourant les arcs qui maximise notre score tant qu’il nous reste des sommets « obligatoires » à traverser. Pour cela on utilise un algorithme, présenté à la page suivante.

// Entrée : un graphe\_minimise G et un sommet de départ

// Sortie : une liste de sommets représentant le chemin parcouru

// Initialisation

Créer une liste vide chemin

sommet\_en\_cours = sommet de départ

Ajouter sommet\_en\_cours à chemin

Créer une liste sommets\_a\_parcourir contenant tous les sommets obligatoires

// Boucle principale

Tant que sommets\_a\_parcourir n'est pas vide

Créer un arc\_poids\_max non null bouclé sur lui même

Pour chaque arc dans les arcs de sommet\_en\_cours

Si l'arc a un poids supérieur à arc\_poids\_max et que l'arrivée de l'arc n'est pas déjà dans le chemin

Mettre à jour arc\_poids\_max avec l'arc

Fin Si

Fin pour

Si arc\_poids\_max a un poids >= 0

mettre à jour sommet\_en\_cours avec l'arrivée de arc\_poids\_max

ajouter sommet\_en\_cours à chemin

Si sommet\_en\_cours est dans sommets\_a\_parcourir

supprimer sommet\_en\_cours de sommets\_a\_parcourir

Fin Si

Sinon

Si sommet\_en\_cours est dans sommets\_a\_parcourir

Mettre à jour sommet\_en\_cours avec l'arrivée de arc\_poids\_max

Ajouter sommet\_en\_cours à chemin

Supprimer sommet\_en\_cours de sommets\_a\_parcourir

Pour chaque sommet\_restant dans sommets\_a\_parcourir

Ajouter sommet\_restant à chemin

Supprimer sommet\_restant de sommets\_a\_parcourir

Fin Pour

Sinon

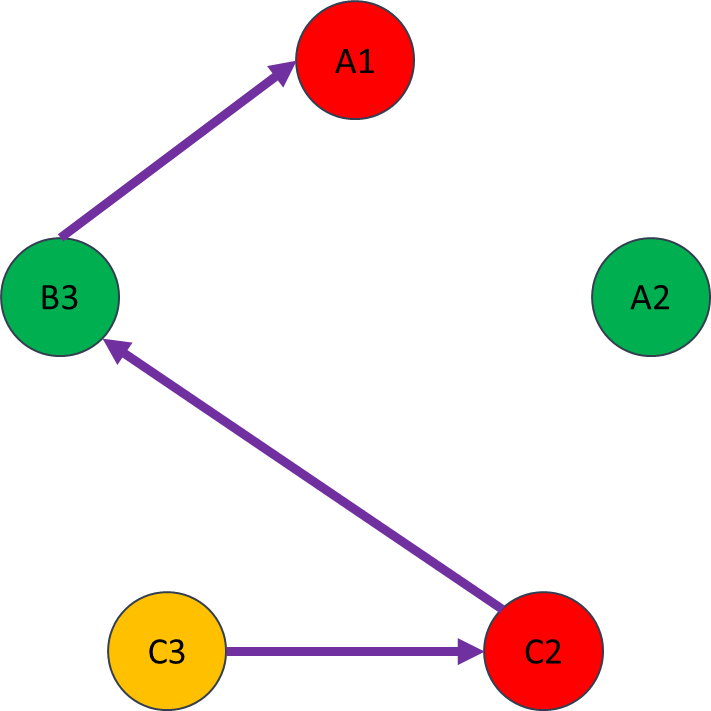
Sortir de la boucle principale

Fin si

Fin si

Fin Tant que

Retourner le chemin



Le chemin obtenu par cette algorithme dans notre graphe complet est maintenant à « développer » pour pouvoir être superposé avec notre graphe initial. Il nous faut alors une fonction qui va rechercher une nouvelle fois le chemin le plus court entre chacun des sommets successif de notre chemin pour obtenir le chemin global final.

// Entrée : une liste de sommets représentant un chemin dans un graphe pondéré et connexe

// Sortie : une liste de sommets représentant un chemin développé par la recherche du plus court chemin entre chaque paire de sommets consécutifs

// Initialisation

Créer une liste vide chemin\_developpe

Créer une liste vide chemin\_tmp

// Ajout du premier sommet du chemin dans la liste développée

Ajouter le premier sommet de la liste chemin dans la liste chemin\_developpe

// Boucle principale

Pour chaque paire de sommets consécutifs sommet\_A et sommet\_B du chemin

Appliquer l'algorithme de Dijkstra entre sommet\_A et sommet\_B pour obtenir le plus court chemin chemin\_tmp

Supprimer le premier sommet de chemin\_tmp car il est déjà présent dans la liste chemin\_developpe

Ajouter tous les sommets de chemin\_tmp dans la liste chemin\_developpe

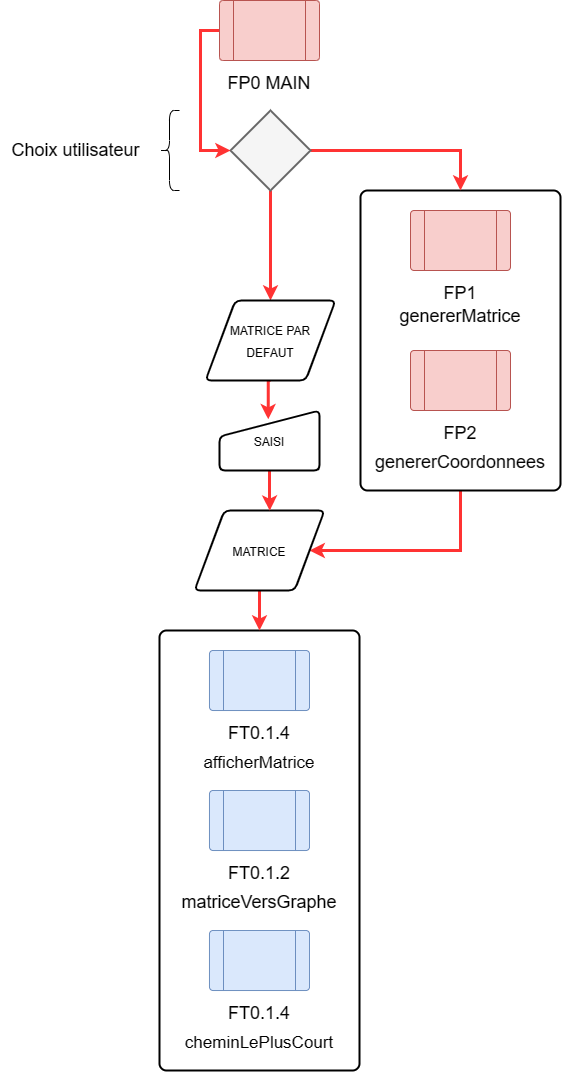
Fin pour

// Ajout du dernier sommet du chemin dans la liste développée

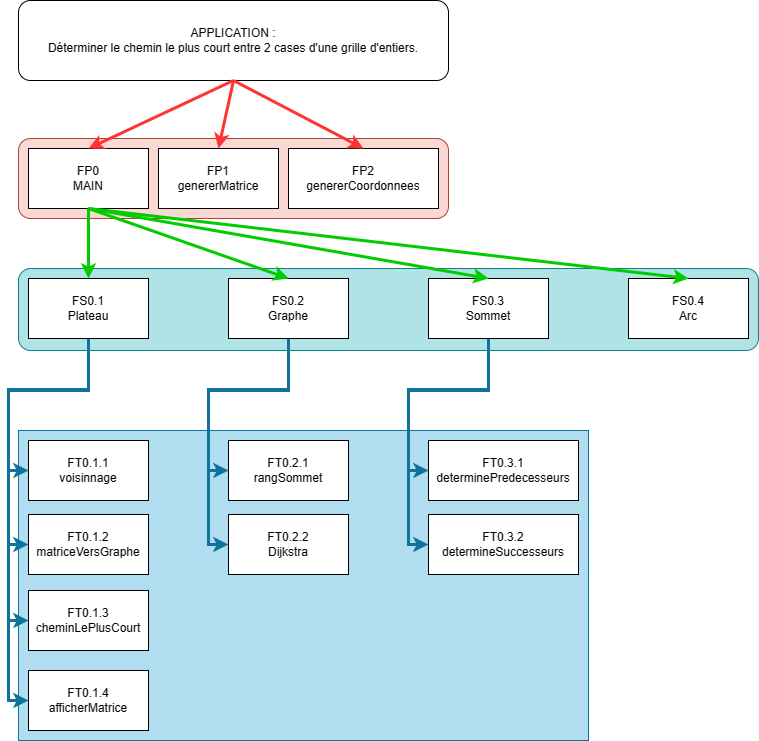
ajouter le dernier sommet de la liste chemin dans la liste chemin\_developpe

Retourner le chemin\_developpe

**Logigramme fonctionnel**



**Arbre hiérarchique**



**Analyse de niveau 1**

|  |  |
| --- | --- |
| FP0  MAIN |  |
| **Valeur Ajoutée**  Cette fonction principale permet de lancer le programme. Elle permet de choisir entre un plateau de jeu aléatoire ou un plateau de jeu prédéfini. On y utilise l'algo de Dijkstra pour trouver le chemin le plus court entre deux points. | |
| **INPUT**  …………… | **OUTPUT**  …………… |
| **Logigramme Fonctionnel**  MAIN | |
| **Service Fonctionnel**  Début du programme  Créer une matrice de 3x3 avec des valeurs entières prédéfinies.  Demander à l'utilisateur de choisir entre deux options en affichant un menu :  1.Trouver le chemin le plus court pour la matrice prédéfinie.  2.Trouver le chemin le plus court pour une matrice aléatoire générée par le programme.  Si l'utilisateur choisit l'option 1 :  Demander à l'utilisateur de saisir deux couples de coordonnées : une pour la case de départ et une pour la case d'arrivée.  Créer un objet Plateau à partir de la matrice.  (FT0.1.2) Transformer la matrice en graphe.  (FT0.1.4) Afficher la matrice.  (FT0.1.3) Trouver le chemin le plus court entre les coordonnées de départ et d'arrivée.  FinSi  Si l'utilisateur choisit l'option 2 :  Générer aléatoirement une matrice de dimensions comprises entre 3 et 32 avec des valeurs entières comprises entre 1 et la largeur.  Générer aléatoirement deux couples de coordonnées.  Créer un objet Plateau à partir de la matrice aléatoire.  (FT0.1.2) Transformer la matrice en graphe.  (FT0.1.4) Afficher la matrice.  (FT0.1.3) Trouver le chemin le plus court entre les coordonnées de départ et d'arrivée.  FinSi  Si l'utilisateur ne choisit aucune des deux options, afficher "Choix invalide".  Fin du programme | |

|  |  |
| --- | --- |
| FP1 genererMatrice |  |
| **Valeur Ajoutée**  Cette fonction permet de générer une matrice pour le plateau de jeu. | |
| **INPUT**  Largeur, hauteur  min valeur, max valeur | **OUTPUT**  matrice de taille hauteur x largeur avec des valeurs comprises entre min et max |
| **Logigramme Fonctionnel** | |
| **Service Fonctionnel**  Début de fonction genererMatrice(paramètres largeur, hauteur, min, max)  Déclarer une matrice de taille hauteur x largeur.    Pour chaque chaque ligne de la matrice :  Pour chaque chaque colonne de la matrice :  Affecter à chaque case de la matrice une valeur aléatoire comprise entre min et max.  Fin  Fin    Retourner la matrice générée.  Fin de fonction | |

|  |  |
| --- | --- |
| FP2 genererCoordonnees |  |
| **Valeur Ajoutée**  Cette fonction permet de générer des coordonnées aléatoires de départ et d'arrivée dans le plateau de jeu. | |
| **INPUT**  largeur, hauteur | **OUTPUT**  tableau de coordonnées comprises dans une taille donnée. |
| **Logigramme Fonctionnel** | |
| **Service Fonctionnel**  //FP2 genererCoordonnees  Début de fonction genererCoordonnees(paramètres largeur, hauteur)  Déclarer un tableau de deux entiers qui représente les coordonnées.    Affecter à la première case du tableau une valeur aléatoire comprise entre 0 et hauteur-1.  Affecter à la deuxième case du tableau une valeur aléatoire comprise entre 0 et largeur-1.    Retourner le tableau de coordonnées généré.  Fin de fonction | |

**Analyse de niveau 2**

|  |  |
| --- | --- |
| FS0.1  Plateau |  |
| **Valeur Ajoutée**  Cette classe représente le plateau de jeu. | |
| **INPUT**  …………… | **OUTPUT**  …………… |
| **Logigramme Fonctionnel**  Plateau | |
| **Service Fonctionnel**  //FS0.1  Attributs :  matrice d'entiers : matrice  objet graphe : gaphe  Début constructeur de Plateau (Paramètre matrice)  Attribut matrice = Paramètre matrice  Attribut graphe = Nouvel Objet Graphe.  Fin du constructeur | |

|  |  |
| --- | --- |
| FS0.2  Graphe |  |
| **Valeur Ajoutée**  Cette classe représente un graphe. Un graphe est caractérisé par une liste de sommets.  (Note : le graphe est celui représentant le plateau de jeu, il est pondéré de poids positifs et orienté) | |
| **INPUT**  …………… | **OUTPUT**  …………… |
| **Logigramme Fonctionnel**  Graphe | |
| **Service Fonctionnel**  //FS0.2  liste d'objets Sommet : sommets  Début constructeur de Graphe  Attribut sommet = Nouvelle liste vide  Fin du constructeur | |

|  |  |
| --- | --- |
| FS0.3  Sommet |  |
| **Valeur Ajoutée**  Cette classe représente un sommet du graphe. Un sommet est caractérisé par son nom, ses prédécesseurs, ses successeurs et ses arcs. | |
| **INPUT**  …………… | **OUTPUT**  …………… |
| **Logigramme Fonctionnel**  Graphe | |
| **Service Fonctionnel**  //FS0.3  Attributs :  Une chaine de caractères : nom  Liste de Sommet : predecesseurs  Liste de Sommet : successeurs  Liste d'Arcs : arcs  Constructeur de Sommet (paramètre chaine de caractères nom)  Attribut nom = paramètre nom  Attribut predecesseurs = Nouvelle liste vide  Attribut successeurs = Nouvelle liste vide  Attribut arcs = nouvelle liste vide  Fin de constructeur de Sommet  Constructeur de Sommet (paramètre Sommet sommet)  Attribut nom = paramètre sommet->nom  Attribut predecesseurs = sommet->predecesseurs  Attribut successeurs = sommet->successeurs  Attribut arcs = sommet->arcs  Fin de constructeur de Sommet | |

|  |  |
| --- | --- |
| FS0.4  Arc |  |
| **Valeur Ajoutée**  Cette classe représente un arc du graphe. Un arc est caractérisé par son sommet de départ, son sommet d'arrivée et son poids. | |
| **INPUT**  …………… | **OUTPUT**  …………… |
| **Logigramme Fonctionnel**  Arc | |
| **Service Fonctionnel**  //FS0.4  Attributs :  objet sommet : depart  objet sommet : arrive  entier : poids  Constructeur de Arc(paramètres Sommet depart, Sommet arrive, entier poids)  Attribut depart = paramètre depart  Attribut arrive = paramètre arrive  Attribut poids = paramètre poids  Fin de constructeur | |