Université Paris - Dauphine

Master 1 MIAGE

Rapport de projet

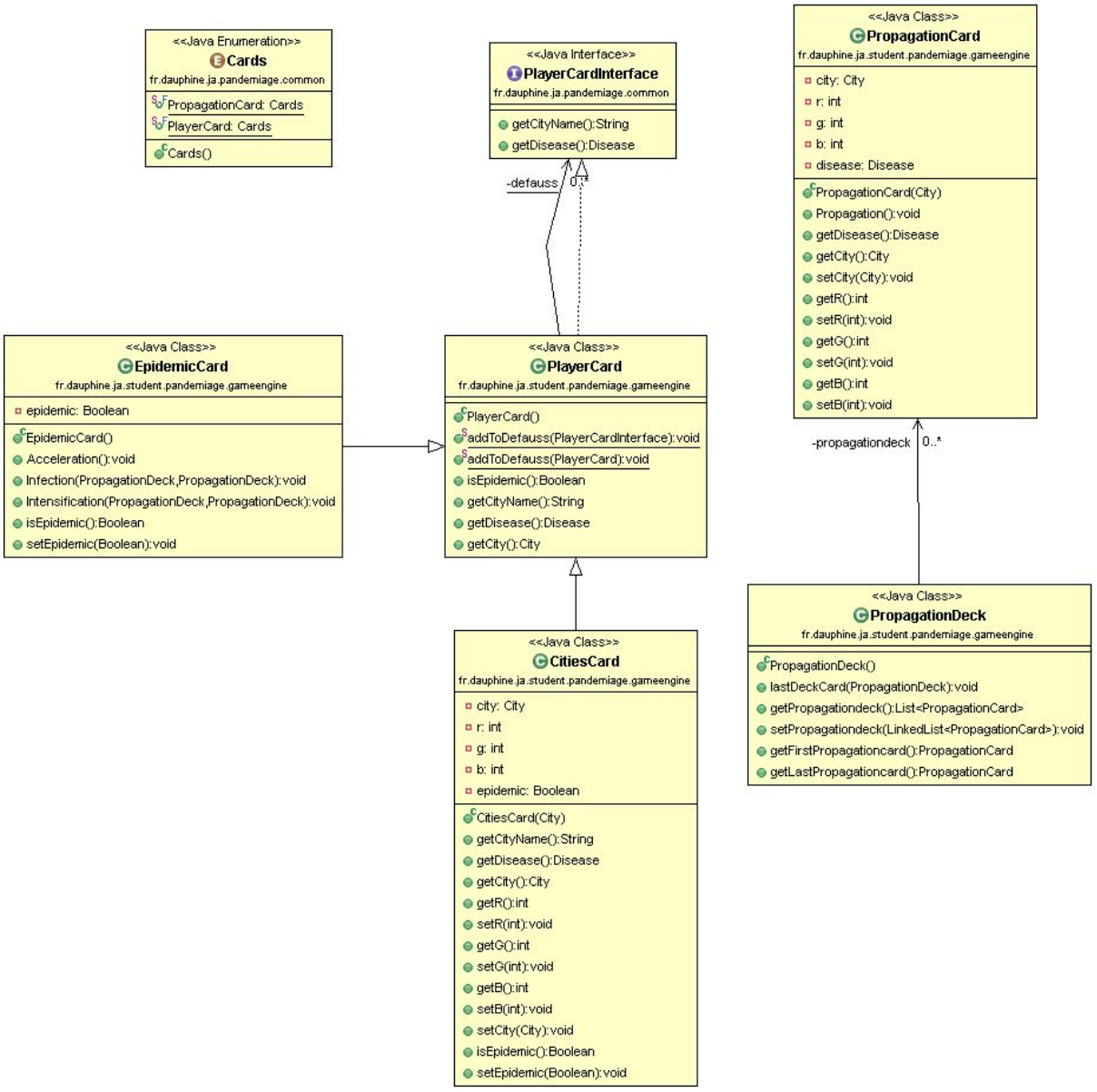
Auteurs :
DERRADJ Akli
ABDELKAFI Imed
ZRIBI Alexandre

Encadrant :
Benjamin Negrevergne

Les règles de jeu données en annexe du projet ne permettant pas une compréhension complète des subtilités du jeu, on s'est référé à la vidéo suivante : https://www.youtube.com/watch?v=7XXMB8UTA50&t=1999s pour éclaircir un certain nombre de point notamment les différences entre types de cartes, le principe d'éclosion ou bien les marqueurs.
Une fois les fondamentaux du jeu acquis, on a fait une première répartition des tâches.
Akli Derradj s'est chargé d'implémenter les classes Player et City.
Imed Abdelkafi s'est occupé de la partie lecture de GraphML ainsi qu'une liste de Villes générée par le parcours du graphe.
Alexandre Zribi a mis en place les classes Card et l'initialisation du moteur de jeu, et tour.

Une fois ces classes finies, on avait un moteur de jeu qui permet de tester l'IA.
L'implémentation de l'IA a été faite en concertation avec tous les membres du groupe.
Les tâches de création des Jars, de gestion des dépendances et le rapport ont aussi été fait en groupe.

Pour la lecture des fichiers GraphML, on a utilisé la librairie Blueprints :
https://github.com/tinkerpop/blueprints.git
Cette librairie permet d'itérer sur les noeuds (*vertices)* qui représentent les villes et ainsi récupérer leurs attributs puis de boucler sur les arêtes (*edges)* et de définir chaque noeud au bout d'une arêtes comme le voisin du noeud à l'autre bout. Une arraylist est ainsi créée pour chaque ville et pourvu avec les voisins de cette dernière.
Aussi une balise *dependency* a été ajoutée dans le fichier pom.xml pour permettre l'exécution dans un environnement quelconque.
La liste des villes est une élément central dans la conception, elle permet d'avoir le statut du jeu et d'exploiter les informations qu'elle contient, elle est constamment mise à jour par les évènements du jeu (tirage de carte épidémie …).



Une réflexion a eu lieu au niveau de la conception des cartes de notre jeu.
En effet, les dépendances entre les types de cartes ne paraissait pas évidentes, et après concertation, la conception suivante a été retenue :

Deux grands types de cartes existent, les PropagationCards et les PlayerCards implémentant PlayerCardInterface.

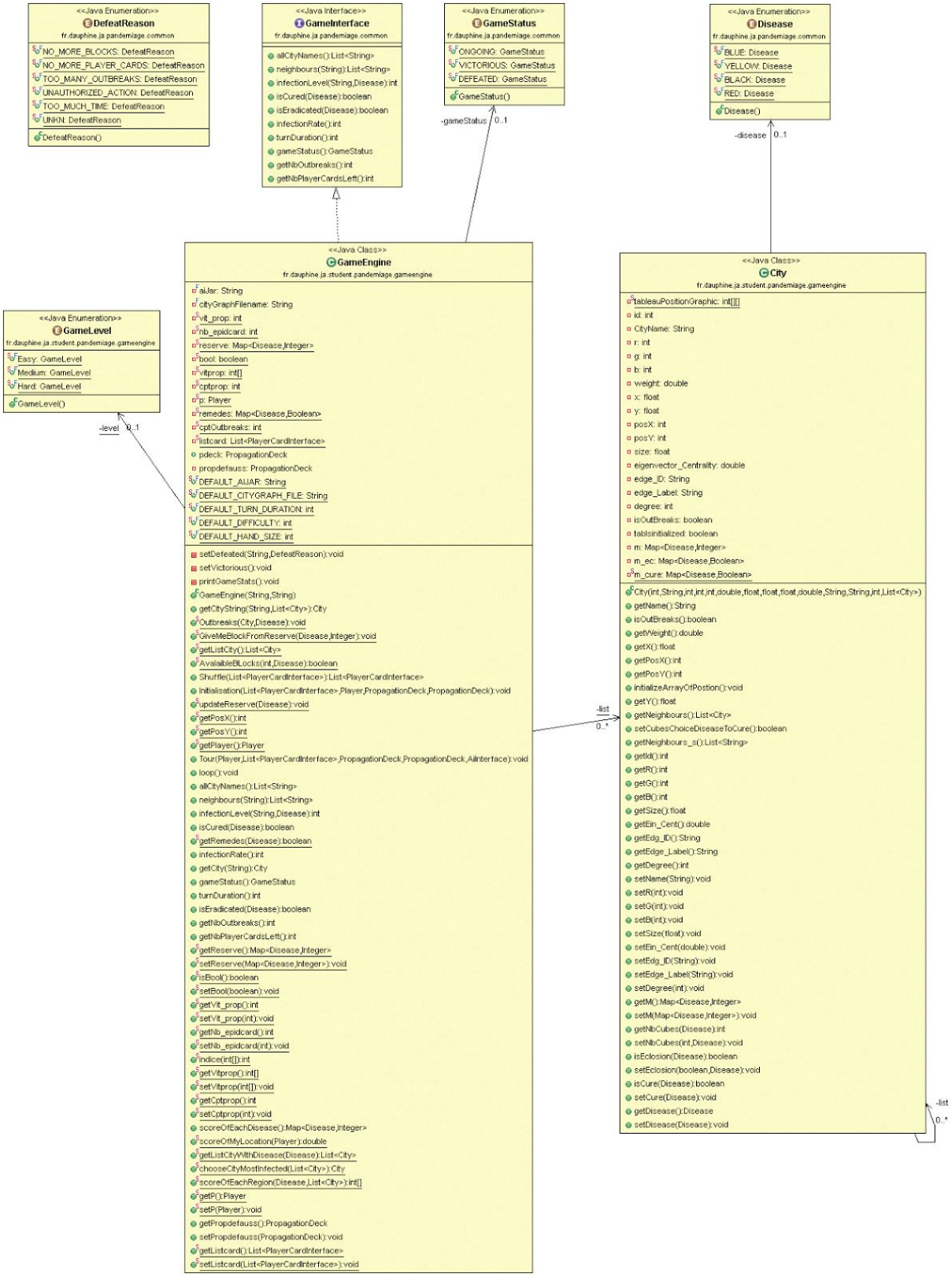
Deux nouvelles classes CitiesCard et EpidemicCard sont créés, héritant de PlayerCard.
Elles sont distinctes car il existe des différences entre les cartes villes et les cartes épidémies, ces dernières n'ayant entre autres pas de villes en attribut ni de maladie.

Cependant ces deux classes héritent toutefois toutes les deux de PlayerCard car une pioche de cartes joueurs, composé de cartes villes et de cartes épidémies est créé.

Les cartes propagations sont quant à elles différenciées des cartes joueurs et possèdent donc une classe à part.

Un objet PlayerCard possède un attribut List<PlayerCardInterface> défauss qui est statique, ce pour avoir la défausse qui est partagée entre toutes les cartes joueurs.
La pioche quant à elle a été définie dans GameEngine.
La défausse et la pioche des cartes propagations sont également définies dans GameEngine.
Les méthodes qui sont issues de l'implémentation de PlayerCardInterface dans PlayerCard sont redéfinies dans la classe CitiesCard.

Moteur de Jeu


«Java Enumeration»
DefeatReason
fr.dauphine.ja.pandemiage.common
NO_MORE_BLOCKS: DefeatReason
NO_MORE_PLAYER_CARDS: DefeatReason
TOO_MANY_OUTBREAKS: DefeatReason
UNAUTHORIZED_ACTION: DefeatReason
TOO_MUCH_TIME: DefeatReason
UNKN: DefeatReason
DefeatReason()
«Java Interface»
GameInterface
fr.dauphine.ja.pandemiage.common
allCityNames():List<String>
neighbours(String):List<String>
infectionLevel(String,Disease):int
isCured(Disease):boolean
isEradicated(Disease):boolean
infectionRate():int
turnDuration():int
gameStatus():GameStatus
getNbOutbreaks():int
getNbPlayerCardsLeft():int
«Java Enumeration»
GameStatus
fr.dauphine.ja.pandemiage.common
ONGOING: GameStatus
VICTORIOUS: GameStatus
DEFEATED: GameStatus
GameStatus()
-gameStatus
0..1
«Java Enumeration»
Disease
fr.dauphine.ja.pandemiage.common
BLUE: Disease
YELLOW: Disease
BLACK: Disease
RED: Disease
Disease()
-disease
0..1
«Java Enumeration»
GameLevel
fr.dauphine.ja.student.pandemiage.gameengine
Easy: GameLevel
Medium: GameLevel
Hard: GameLevel
GameLevel()
-level
0..1
«Java Class»
GameEngine
fr.dauphine.ja.student.pandemiage.gameengine
aiJar: String
cityGraphFilename: String
vit_prop: int
nb_epidcard: int
reserve: Map<Disease,Integer>
bool: boolean
vitprop: int[]
cptprop: int
p: Player
remedes: Map<Disease,Boolean>
cptOutbreaks: int
listcard: List<PlayerCardInterface>
pdeck: PropagationDeck
propdefauss: PropagationDeck
DEFAULT_AIJAR: String
DEFAULT_CITYGRAPH_FILE: String
DEFAULT_TURN_DURATION: int
DEFAULT_DIFFICULTY: int
DEFAULT_HAND_SIZE: int
setDefeated(String,DefeatReason):void
setVictorious():void
printGameStats():void
GameEngine(String,String)
getCityString(String,List<City>):City
Outbreaks(City,Disease):void
GiveMeBlockFromReserve(Disease,Integer):void
getListCity():List<City>
AvalaibleBLocks(int,Disease):boolean
Shuffle(List<PlayerCardInterface>):List<PlayerCardInterface>
Initialisation(List<PlayerCardInterface>,Player,PropagationDeck,PropagationDeck):void
updateReserve(Disease):void
getPosX():int
getPosY():int
getPlayer():Player
Tour(Player,List<PlayerCardInterface>,PropagationDeck,PropagationDeck,AiInterface):void
loop():void
allCityNames():List<String>
neighbours(String):List<String>
infectionLevel(String,Disease):int
isCured(Disease):boolean
getRemedes(Disease):boolean
infectionRate():int
getCity(String):City
gameStatus():GameStatus
turnDuration():int
isEradicated(Disease):boolean
getNbOutbreaks():int
getNbPlayerCardsLeft():int
getReserve():Map<Disease,Integer>
setReserve(Map<Disease,Integer>):void
isBool():boolean
setBool(boolean):void
getVit_prop():int
setVit_prop(int):void
getNb_epidcard():int
setNb_epidcard(int):void
indice(int[]):int
getVitprop():int[]
setVitprop(int[]):void
getCptprop():int
setCptprop(int):void
scoreOfEachDisease():Map<Disease,Integer>
scoreOfMyLocation(Player):double
getListCityWithDisease(Disease):List<City>
chooseCityMostInfected(List<City>):City
scoreOfEachRegion(Disease,List<City>):int[]
getP():Player
setP(Player):void
getPropdefauss():PropagationDeck
setPropdefauss(PropagationDeck):void
getListcard():List<PlayerCardInterface>
setListcard(List<PlayerCardInterface>):void
«Java Class»
City
fr.dauphine.ja.student.pandemiage.gameengine
tableauPositionGraphic: int[][]
id: int
CityName: String
r: int
g: int
b: int
weight: double
x: float
y: float
posX: int
posY: int
size: float
eigenvector_Centrality: double
edge_ID: String
edge_Label: String
degree: int
isOutBreaks: boolean
tabIsinitialized: boolean
m: Map<Disease,Integer>
m_ec: Map<Disease,Boolean>
m_cure: Map<Disease,Boolean>
City(int,String,int,int,int,double,float,float,float,double,String,String,int,List<City>)
getName():String
isOutBreaks():boolean
getWeight():double
getX():float
getPosX():int
getPosY():int
initializeArrayOfPostion():void
getY():float
getNeighbours():List<City>
setCubesChoiceDiseaseToCure():boolean
getNeighbours_s():List<String>
getId():int
getR():int
getG():int
getB():int
getSize():float
getEin_Cent():double
getEdg_ID():String
getEdge_Label():String
getDegree():int
setName(String):void
setR(int):void
setG(int):void
setB(int):void
setSize(float):void
setEin_Cent(double):void
setEdg_ID(String):void
setEdge_Label(String):void
setDegree(int):void
getM():Map<Disease,Integer>
setM(Map<Disease,Integer>):void
getNbCubes(Disease):int
setNbCubes(int,Disease):void
isEclosion(Disease):boolean
setEclosion(boolean,Disease):void
isCure(Disease):boolean
setCure(Disease):void
getDisease():Disease
setDisease(Disease):void
-list
0..*
-list
0..*

Au niveau du moteur de jeu, la méthode Initialisation consiste à simuler les premiers tours où on ne tire que des cartes propagation.

Selon le niveau de difficulté, on a un nombre de piles créés égal au nombre de cartes épidémies dispatchés dans ces dernières.

Ensuite on crée une grande pile *listcard* qui sera celle dans laquelle on va piocher durant le jeu.

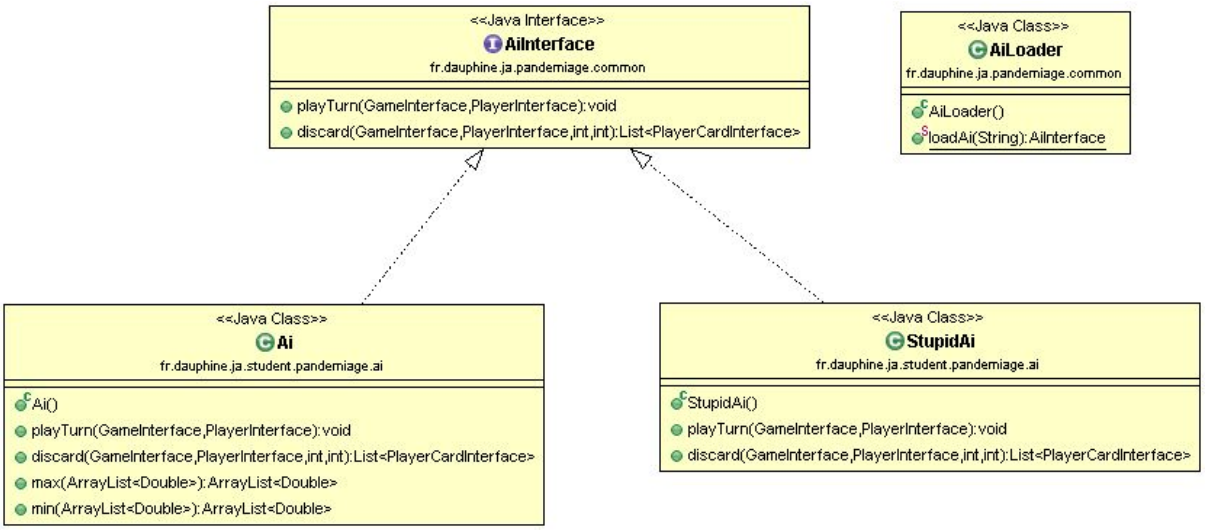
Ensuite la méthode Tour définit tout ce qui se déroule lors d'un tour (on pioche 2 cartes joueurs etc…) et les conditions où l'utilisateur gagne ou perd.
L'exécution de notre méthode ne s'arrête que lorsque le statut de notre jeu; on concède une défaite ou bien on gagne la partie.

Une fenêtre dont le contenu est régulièrement mis à jour a été créée dans la méthode Tour.

Une classe permettant de définir le contenu de notre fenêtre, qui servira d'interface est donc implémentée.

Dans celle-ci, les Awt et Swing sont exploitées.
Des images à fond transparent sont implémentées afin d'avoir un rendu convenable.
Le contenu de la fenêtre est mis à jour en utilisant la méthode Repaint().

Intelligence Artificielle



On a décidé d'utiliser une méthode usant de différentes heuristiques qui seront ensuite regroupées, selon des poids prédéfinis. Ces poids ont été définis par nos soins sur la base des résultats obtenus sur plusieurs tests.
En ce qui concerne les heuristiques, on a mis en place plusieurs critères de sélection pour les cartes et les actions à effectuer.
Voici l'énumération de ces critères :

- En ce qui concerne les cartes joueurs, qui est utilisé dans « playturn et discard », elles seront notées sur plusieurs critères que voici :
    1. La pertinence de la zone où la carte peut m'emmener, grâce aux méthodes « FlyTo, FlyToCharter ». C'est-à-dire, si la ville sur la carte me permet d'aller vers une région qui est plus infecté que la région où je me situe, alors je choisis d'utiliser cette carte avec un FlyTo.

    Autre exemple, si la ville sur la carte est la même que ma position courante, alors cette carte aura un très gros score car elle me permettra d'aller où je veux.
    2. La capacité que cette carte a à me permettre de me procurer un remède. Ce score est calculé sur la base du nombre de cartes que je dispose déjà dans ma main. Si j'ai plusieurs cartes de la même couleur en main, ces cartes auront plus de valeurs.
    3. La capacité de cette carte à m'emmener vers une ville qui est très proche des villes infecté de la région. La différence par rapport au critère 1, réside dans le fait que dans le premier critère on s'intéresse à la région

où cette carte peut m'emmener, tandis que le troisième critère prend également en compte si la ville où je peux aller est proche des villes infectées, ce qui permet de faire un gain en nombre d'action.

Ces trois critères sont ensuite sommés sous cette forme : move+cure+region*3;

· En ce qui concerne le choix des actions « playTurn » : La méthode playTurn se divise en deux grandes parties

On a décidé que l'action prioritaire serait de retirer le maximum de cubes dans ma ville, s'ils existent. Le choix de la maladie « Disease » à retirer se fait selon la méthode setCubesChoiceDiseaseToCure qui choisit parmi les maladies se trouvant à ma ville courantes celle qui a le plus grand nombre de cubes, et lui enlève un cube si on a pas encore le remède, tous les cubes sinon.

1. Calculer le score des cartes que j'ai en main, avec les méthodes définis plus haut.
2. Calculer le score que pourrait rapporter des déplacements autour des voisins de ma ville courante. Ce score est calculé selon le score de la région où je me situe, en utilisant la méthode définit plus haut.

Ensuite un choix est fait entre ces deux critères, on choisit la valeur la plus grande est choisie.

Si le choix se porte sur l'utilisation d'une carte, on fait alors soit un flyto soit un flycharter. Si le flycharter est choisit (la ville sur la carte est la même que ma ville courante) alors le choix de la ville d'arrivée est décidé par *chooseCityMostInfected* qui retrouve la ville la plus infecté de la MAP.

Si le choix se porte sur un déplacement vers mes voisins, alors on effectue un traitement supplémentaire, nous parcourons nos voisins et leurs voisins, nous retrouvons la ville la plus infecté parmi les voisins et nous décidons ensuite d'y aller avec MoveTo.

Ce choix est fait sachant que nous nous trouvons dans la région la plus infectée de la Map.

· En ce qui concerne les cartes à jeter, lorsque le nombre de carte maximum en main est dépassé : Le choix des cartes à jeter se fait selon le score de ces dernières. C'est-à-dire les cartes ayant les scores les plus faibles.

Voici le diagramme de classes de la classe Player :


<<Java Interface>>
PlayerInterface
fr.dauphine.ja.pandemiage.common
moveTo(String):void
flyTo(String):void
flyToCharter(String):void
skipTurn():void
treatDisease(Disease):void
discoverCure(List<PlayerCardInterface>):void
playerLocation():String
playerHand():List<PlayerCardInterface>
<<Java Class>>
Player
fr.dauphine.ja.student.pandemiage.gameengine
listCity: List<City>
listCardHand: List<PlayerCardInterface>
currentCity: City
action: int
switchturn: boolean
Player(City,List<City>)
moveTo(String):void
getCurrentCity():City
getActionLeft():int
flyTo(String):void
flyToCharter(String):void
skipTurn():void
treatDisease(Disease):void
discoverCure(List<PlayerCardInterface>):void
playerLocation():String
playerHand():List<PlayerCardInterface>
SeeCards():void
scoreOfEachCardInRegion(PlayerCardInterface):double
scoreOfEachCardInCure(PlayerCardInterface):int
scoreOfEachCardInMove(PlayerCardInterface):int
scoreOfTheCard(PlayerCardInterface):double
addToPlayerHand(PlayerCardInterface):void
getListCity():List<City>
setListCity(List<City>):void
getListCardHand():List<PlayerCardInterface>
setListCardHand(List<PlayerCardInterface>):void
getAction():int
setAction(int):void
isSwitchturn():boolean
setSwitchturn(boolean):void
setCurrentCity(City):void