lsflw Documentation

Release 1.0

CARON Cyril HOULEGATTE Pauline HOWELL Tuesday PINEAU Ma

January 25, 2015

CONTENTS

Voici la documentation des classes pour la réalisation du mini projet

CHAPTER

ONE

CELLULE

class Cellule.**Cellule**(*numero*, *attaque*, *defense*, *attaqueMax*, *defenseMax*, *production*, *couleur*, *x*, *y*, *rayon*)

Une cellule du jeu. Elle peut être représentée par un sommet dans un graphe.

Parameters

- **numero** (*int*) – le numéro identifiant la cellule (unique pour chaque cellule)
- **attaque** (*int*) – le nombre d'unités attaquantes sur la cellule actuellement
- **defense** (*int*) – le nombre d'unités defensives sur la cellule actuellement
- **attaqueMax** (*int*) – le nombre d'unités attaquantes maximal sur la cellule
- **defenseMax** (*int*) – le nombre d'unités defensives maximal sur la cellule
- **production** (*int*) – vitesse de production des unités de la cellule
- **couleurJoueur** (*int*) – couleur du joueur à qui appartient la cellule (-1 si neutre)
- **x** (*int*) – la coordonnée x de la cellule sur le terrain (graphique)
- **y** (*int*) – la coordonnée y de la cellule sur le terrain (graphique)
- **rayon** (*int*) – le rayon de la cellule (graphique)

Raises

- **CelluleException** – si l'attaque est inférieure à 0 ou supérieure à l'attaque maximale
- **CelluleException** – si la défense est inférieure à 0 ou supérieure à la défense maximale

aPourCouleur(*couleur*)

Retourne vrai si la cellule possède la couleur passée en paramètre, et faux sinon. Autrement dit, elle retourne vrai si la cellule appartient au joueur ayant la couleur passée en paramètre.

Parameters **couleur** (*int*) – la couleur à vérifier

Returns retourne vrai si la cellule a bien pour couleur la couleur donnée en paramètre, faux sinon.

Return type boolean

ajouterLien(*lien*)

Ajoute un lien reliant cette cellule à une autre.

Parameters **lien** (*Lien*) – le lien à ajouter

Raises **CelluleException** si cette cellule n'est pas l'une des cellules aux extrémités du lien.

getAttaque()

Retourne l'attaque actuelle de la cellule.

Returns l'attaque de la cellule

Return type int

getAttaqueMax()
Retourne la quantité maximale d'unités offensives que la cellule peut contenir

Returns l'attaque maximale de la cellule

Return type int

getCouleur()
Retourne la couleur de la cellule. Cette couleur correspond à un numéro de joueur supérieur ou égale à -1 (-1 étant la couleur du neutre). Cette couleur permet de définir à quel joueur appartient la cellule.

Returns la couleur de la cellule

Return type int

getCout()
Retourne le coût de base de la cellule pour être conquise, c'est à dire le nombre d'unités nécéssaire pour la capturer (attaque + défense). On ne prend pas en compte les unités présentes sur les liens.

Returns le coût de la cellule

Return type int

getDefense()
Retourne la défense actuelle de la cellule.

Returns la défense de la cellule

Return type int

getDefenseMax()
Retourne la quantité maximale d'unités défensives que la cellule peut contenir

Returns la défense maximale de la cellule

Return type int

getExcedent()
Retourne l'excédent de la cellule. L'excédent correspond au trop plein d'unités que la cellule va produire ou recevoir, et perdre (car elle aura déjà atteint son attaque maximale). Seules les unités des mouvements arrivant vers cette cellule avec un temps restant inférieur à 1000 sont prises en comptes.

Returns le nombre d'unités excédent

Return type int

getLiens()
Retourne les liens dont cette cellule est l'une des extrémités.

Returns la liste des liens

Return type List<Lien>

getMouvementsVersCellule()
Méthode qui retourne la liste de tous les mouvements allant vers cette cellule.

Returns la liste de tous les mouvements allant vers cette cellule.

Return type List<Mouvement>

getNumero()
Retourne le numéro unique identifiant la cellule

Returns le numéro de la cellule

Return type int

getPourcentageAttaque()
Retourne à combien de pourcentage la cellule est pleine. 1 si l'attaque de la cellule est égale à l'attaque maximale.

Returns le pourcentage de l'attaque de la cellule

Return type int

getProduction()
Retourne la valeur de la cadence de production de la cellule

Returns la cadence de production de la cellule

Return type int

getVoisins()
Retourne la liste des cellules voisines à celle-ci.

ATTENTION, peut renvoyer un résultat différent de la méthode Terrain.getVoisinsCellule(cellule) car on ne modifiera jamais la liste des liens de la cellule, contrairement au terrain, ou l'on pourra ne considérer qu'une partie du terrain (un sous graphe), et donc qu'une partie des liens.

Returns la liste des cellules voisines

Return type List<Cellule>

getVoisinsAllies()
Retourne la liste des cellules alliées de cette cellule, c'est à dire celles qui ont la même couleur que celle de cette cellule.

Returns la liste des cellules alliées de cette cellule

Return type List<Cellule>

getVoisinsEnnemis()
Retourne la liste des cellules ennemies de cette cellule, c'est à dire celles qui n'ont pas la même couleur que celle de cette cellule.

Returns la liste des cellules ennemies de cette cellule

Return type List<Cellule>

setAttaque(*attaque*)
Affecte la valeur de la variable attaque actuelle. La nouvelle valeur de l'attaque ne peut pas être inférieure à 0, ni excéder l'attaque maximale de la cellule.

Parameters **attaque** (*int*) – la nouvelle valeur de l'attaque actuelle de la cellule

Raises **CelluleException** si l'attaque entrée en paramètre est inférieure à 0 ou supérieure à l'attaque maximale

setCouleur(*couleur*)
Affecte la valeur de la variable couleur (la cellule change de propriétaire). Cette couleur doit être supérieure ou égale à -1, -1 étant la couleur du joueur neutre.

Parameters **couleur** (*int*) – la nouvelle couleur de la cellule.

Raises **CelluleException** si la couleur n'est pas supérieure ou égale à -1.

setDefense(*defense*)
Affecte la valeur de la variable defense actuelle. La nouvelle valeur de la défense ne peut pas être inférieure à 0, ni excéder la défense maximale de la cellule.

Parameters **defense** (*int*) – la nouvelle valeur de la défense actuelle de la cellule

Raises **CelluleException** si la défense entrée en paramètre est inférieure à 0 ou supérieure à la défense maximale

toString()

Représente l'objet sous forme de chaine

Returns retourne une chaine de caractère représentant la cellule

Return type str

vaEtrePrise()

vérifie si la cellule va être conquise par un autre joueur (elle va changer de propriétaire). Si c'est le cas, elle retourne True, False sinon.

Returns retourne vrai si cette cellule va être conquise par un autre joueur, faux sinon

Return type boolean

CHAPTER

TWO

LIEN

class Lien.**Lien**(*u*, *v*, *distance*)

Les liens reliant les cellules du terrain (représentent les arêtes du graphe).

On mettra toujours la cellule ayant le plus petit numéro dans l'attribut u.

Parameters

- **u** (*Cellule*) – Une cellule de l'une des extrémités du lien
- **v** (*Cellule*) – L'autre cellule de l'une des extrémités du lien
- **distance** (*int*) – la distance séparant les cellules u et v

Raises LienException si la distance est inférieure ou égale à 0

ajouterMouvementVersCellule(*cellule*, *mouvement*)

Ajoute le mouvement passé en paramètre vers la cellule passée en paramètre

Parameters

- **cellule** (*Cellule*) – La cellule à laquelle on ajoute le mouvement
- **mouvement** (*Mouvement*) – Le mouvement à ajouter

Raises LienException si la cellule n'appartient pas au lien

ajouterMouvementVersU(*mouvement*)

Ajoute un mouvement vers la cellule enregistrée sous U

Parameters **mouvement** (*Mouvement*) – Le mouvement à ajouter

ajouterMouvementVersV(*mouvement*)

Ajoute un mouvement vers la cellule enregistrée sous V

Parameters **mouvement** (*Mouvement*) – Le mouvement à ajouter.

celluleAppartientAuLien(*cellule*)

Retourne vrai si la cellule passée en paramètre appartient au lien, faux sinon.

Parameters **cellule** (*Cellule*) – Cellule dont on veut savoir si elle appartient au lien

Return type booleen

clearAllMouvements()

Supprime tous les mouvements présents sur le lien.

getDistance()

Retourne la longueur du lien, c'est à dire la distance séparant les deux cellules aux extrémités du lien.

Returns la longueur du lien

Return type int

getMouvementsVersCellule(*cellule*)
Retourne la liste des mouvements allant vers la cellule passée en paramètre.

Parameters **cellule** (*Cellule*) – La cellule dont on veut récupérer les mouvements entrants

Returns la liste des mouvements allant vers cette cellule

Return type List<Mouvement>

Raises LienException si la cellule n'appartient pas au lien

getMouvementsVersU()
Retourne la liste des mouvements allant vers U

Returns la liste des mouvements allant vers U

Return type List<Mouvement>

getMouvementsVersV()
Retourne la liste des mouvements allant vers V

Returns la liste des mouvements allant vers V

Return type List<Mouvement>

getOtherCellule(*cellule_inconnue*)
Selon la cellule donnée en paramètre, retourne l'autre cellule du lien (retourne U si on donne V et vice versa).

Raises LienException si la cellule inconnue n'appartient pas au lien

Returns l'autre cellule du lien

Return type Cellule

getU()
Retourne la cellule enregistrée sous l'attribut U

Returns la cellule enregistrée dans U

Return type Cellule

getV()
Retourne la cellule enregistrée sous l'attribut V

Returns la cellule enregistrée dans V

Return type Cellule

hachage(*cellule1*, *cellule2*)
Permet de calculer la valeur unique qui identifie un lien supposé entre deux cellules, Retourne la valeur du hash d'un lien à partir de ces deux cellules.

Parameters

- **cellule1** (*Cellule*) – la première cellule
- **cellule2** (*Cellule*) – la deuxième cellule

Returns l'identifiant du lien supposé entre les deux cellules

Return type int

hash()
Utilisée pour ranger les liens dans un dictionnaire Retourne la valeur unique qui identifie le lien

Returns la valeur unique identifiant le lien.

Return type int

toString()

Retourne des informations textuelles sur le lien

Returns le lien sous forme d'une chaine de caractère

Return type str

CHAPTER

THREE

MOUVEMENT

class Mouvement.**Mouvement**(*depuis*, *vers*, *nbUnites*, *couleurJoueur*, *distance*, *vitesse*, *temps_depart*, *temps_actuel*)

Un mouvement représente un déplacement d'unités sur un lien.

Parameters

- **depuis** (*Cellule*) – la cellule au départ du mouvement
- **vers** (*Cellule*) – la cellule à l'arrivée du mouvement
- **nbUnites** (*int*) – le nombre d'unités qui sont sur le mouvement
- **couleurJoueur** (*int*) – le numéro du joueur auquel appartiennent les unités offensives
- **distance** (*int*) – la distance à parcourir sur le lien
- **temps_depart** (*int*) – temps du serveur lors de l'envoi du mouvement
- **temps_actuel** (*int*) – temps du serveur

aPourCouleur(*couleurJoueur*)

Retourne vrai si le mouvement possède la couleur passée en paramètre. (donc si le mouvement appartient au joueur ayant cette couleur)

Parameters **couleurJoueur** (*int*) – la couleur du joueur

Returns vrai si le mouvement possède cette couleur, faux sinon.

Return type booleen

fromCellule()

Retourne la cellule depuis laquelle le mouvement est originaire

Returns la cellule depuis laquelle le mouvement est originaire

Return type Cellule

getCouleur()

Retourne la couleur du mouvement, c'est à dire le numéro du joueur qui envoie le mouvement.

Returns la couleur du mouvement

Return type int

getDistance()

Retourne la distance totale à parcourir par le mouvement.

Returns la distance totale à parcourir par le mouvement

Return type int

getNbUnites()
Retourne le nombre d'unités sur le mouvement

Returns le nombre d'unités

Return type int

getTempsActuel()
Retourne le temps actuel du serveur

Returns le temps actuel du serveur

Return type int

getTempsDepart()
Retourne le temps de départ du mouvement

Returns le temps de départ du mouvement

Return type int

getTempsRestant()
Retourne le temps restant à parcourir avant l'arrivée du mouvement à destination.

Returns le temps restant à parcourir

Return type int

getVitesse()
Retourne la vitesse du mouvement

Returns la vitesse du mouvement

Return type int

setTempsActuel(*temps_actuel*)
Affecte la variable temps_actuel avec la valeur passée en paramètre

Parameters **temps_actuel** (*int*) – le temps du serveur

toCellule()
Retourne la cellule vers laquelle le mouvement se dirige

Returns la cellule vers laquelle le mouvement se dirige

Return type Cellule

toOrder(*uid*)
Retourne l'ordre correspondant au mouvement associé dans la forme du protocole du serveur

Returns l'ordre correspondant au mouvement

Return type str

CHAPTER

FOUR

ROBOT

class Robot.**Robot**(*uid*)

La classe Robot. Initialise le robot. A appeler dans la procédure 'register_pooo(uid)'

Parameters **uid** (*str*) – identifiant unique du robot que le serveur lui a attribué

analyseMessage(*state*)

Décompose la chaine state passée en paramètre et agit en conséquence, exécute soit :

- state of game
- game over
- end of game

Parameters **state** (*str*) – la chaine reçue, envoyée par le serveur, peut contenir un message STATE, GAMEOVER, ou ENDOFGAME

end_of_game()

Arrête le match en cours

game_over(*state_game_over*)

L'un des participants du match a perdu, on analyse plus en détail la chaine reçue pour savoir si c'est ce robot qui ne peut plus jouer.

Parameters **state_game_over** (*str*) – chaine de caractère GAMEOVER envoyé par le serveur

getDecisions()

Retourne la liste des décisions, chacune conforme au protocole du serveur. Ces décisions sont prises selon la stratégie adoptée.

exemple : ['[0947e717-02a1-4d83-9470-a941b6e8ed07]100FROM0TO2', '[0947e717-02a1-4d83-9470-a941b6e8ed07]56FROM6TO4']

Returns la liste des décisions

Return type List<str>

getIdMatch()

Retourne l'identifiant du match

Returns l'identifiant du match

Return type str

getMaCouleur()

Retourne la couleur du robot

Returns la couleur du robot

Return type int

getNbJoueurInitial()
Retourne le nombre de joueurs initial

Returns le nombre de joueurs initial

Return type int

getNbJoueurs()
Retourne le nombre de joueurs actuellement en train de jouer

Returns le nombre de joueurs pouvant encore jouer

Return type int

getStrategie()
Retourne la stratégie du robot

Returns la stratégie du robot

Return type Strategie

getTemps()
Retourne le temps actuel du jeu (fournie par le serveur)

Returns le temps actuel du jeu

Return type int

getTerrain()
Retourne le terrain du match en cours

Returns le terrain du match en cours

Return type Terrain

getUID()
Retourne l'identifiant du robot

Returns l'identifiant du robot

Return type str

getVitesse()
Retourne la vitesse du match en cours

Returns la vitesse du match en cours

Return type int

initialiserMatch(*init_string*)
Initialise le robot pour un match. A appeler dans la procédure 'init_pooo(init_string)'

exemple : "INIT20ac18ab-6d18-450e-94af-bee53fdc8fcaTO6[2];1;3CELLS:1(23,9)'2'30'8'I,2(41,55)'1'30'8'II,3(23,103)'

Parameters **init_string** (*str*) – chaîne regroupant les informations envoyées par le serveur pour l'initialisation d'un nouveau match, sous la forme INIT.

partieEnCours()
Retourne vrai si une partie est en cours, faux sinon

Returns vrai si une partie est en cours, faux sinon

Return type booleen

peutJouer()
Retourne vrai si le robot peut jouer, et si il ne peut plus jouer

Returns vrai si le robot a le droit de jouer, faux sinon

Return type booleen

setStrategie(*strategie*)
Modifie la stratégie du robot

Parameters **strategie** (*Strategie*) – la nouvelle stratégie du robot

setTemps(*temps*)
Modifie le temps actuel du jeu, c'est à dire la variable temps du Robot, mais également touts les attributs temps_actuel des mouvements (modifiant ainsi le temps restant).

Parameters **temps** (*int*) – Le temps

updateTerrain(*state*)
Met à jour les informations sur le terrain en fonction de la chaîne passée en paramètre

exemple de chaine state state = "STATE20ac18ab-6d18-450e-94af-bee53fdc8fcaIS2;3CELLS:1[2]12'4,2[2]15'2,3[1]33'6;4MOVES:1<5[2]@232'>6[2]@488'>3[1]@4330'2,1<10[1]@2241'

Parameters **state** (*str*) – la chaîne envoyée par le serveur, de la forme STATE

CHAPTER

FIVE

TERRAIN

class Terrain.**Terrain**

Le terrain du jeu représente un graphe

ajouterCellule(*cellule*)

Ajoute la cellule passée en paramètre dans le terrain

Parameters **cellule** (*Cellule*) – la cellule à ajouter

ajouterLien(*lien*)

Ajoute le lien passé en paramètre dans le terrain

Parameters **lien** (*Lien*) – Le lien à ajouter

Raises **TerrainException** si au moins l'une des cellule du lien n'est pas présente dans le terrain

dijkstra(*depart*, *arrivee*)

Retourne le plus court chemin entre deux cellules passées en paramètre

Parameters

- **depart** (*Cellule*) – La cellule de départ
- **arrivee** (*Cellule*) – La cellule d'arrivée

Returns le plus court chemin entre les deux cellules (une liste de numéro de cellule), et la distance totale à parcourir

Return type (List<int> , int)

getCellule(*numero*)

Retourne la cellule du terrain ayant le numéro associé passé en paramètre

Parameters **numero** (*int*) – Le numéro de la cellule recherchée

Returns la cellule associée au numéro

Return type :Cellule

Raises **TerrainException** si il n'y a aucune cellule dans le terrain possédant ce numéro

getCellules()

Retourne le dictionnaire contenant la liste de toutes les cellules du terrain

Returns le dictionnaire contenant les cellules du terrain

Return type dict

getCellulesJoueur(*couleurJoueur*)

Retourne la liste des cellules appartenant au joueur ayant la couleur passée en paramètre (-1 pour le neutre).

Parameters **couleurJoueur** (*int*) – la couleur du joueur

Returns la liste des cellules de ce joueur

Return type List<Cellule>

getCheminVersCellulePlusProche(*depart*, *arrivees*)
Retourne le chemin depuis une cellule vers la cellule la plus proche selectionnée dans un ensemble

Parameters

- **depart** (*Cellule*) – La cellule de départ
- **arrivees** (*List<Cellules>*) – L'ensemble des cellules d'arrivée

Returns le chemin le plus court entre la cellule de départ et la cellule d'arrivée la plus proche (correspond à la liste des numéro des cellules composant le chemin)

Return type List<int>

getComposantesConnexes()
Retourne la liste des composantes connexes du graphe (le terrain se comporte comme un graphe).

Returns les composantes connexes du graphe.

Return type List<Terrain>

getLien(*numeroLien*)
Retourne le lien du terrain ayant le numéro associé passé en paramètre

Parameters **numeroLien** (*int*) – Le numéro de le lien recherché

Returns le lien associée au numéro

Return type Lien

Raises TerrainException si il n'y a aucun Lien dans le terrain possédant ce numéro

getLienEntreCellules(*cellule_1*, *cellule_2*)
Retourne le lien sur le terrain entre les 2 cellules passées en paramètre

Parameters

- **cellule_1** – La première cellule
- **cellule_2** – La deuxième cellule

Returns le lien entre les deux cellules

Return type Lien

Raises TerrainException si le lien n'est pas présent dans le terrain

getLiens()
Retourne le dictionnaire contenant la liste de touts les liens du terrain

Returns le dictionnaire contenant les liens du terrain

Return type dict

getNbCellules()
Retourne le nombre de cellules présentes dans le terrain

Returns le nombre de cellules sur le terrain

Return type int

getNbLiens()
Retourne le nombre de liens présents dans le terrain

Returns le nombre de liens sur le terrain

Return type int

getSousGraphe(*listeCellules*)
Retourne le sous graphe contenant les cellules données en paramètre

Parameters **listeCellules** (*List<Cellules>*) – La liste des cellules

Returns le sous graphe correspondant

Return type Terrain

getVoisinsCellule(*cellule*)
Retourne la liste des voisins de la cellule sur le terrain passée en paramètre

Parameters **cellule** (*Cellule*) – La cellule dont on veut récupérer la liste des voisins sur le terrain

Returns la liste des cellules voisines à celle passée en paramètre

Return type List<Cellule>

Raises **TerrainException** si la cellule n'est pas présente sur le terrain

toString()
Retourne sous forme de chaine une représentation textuelle du terrain

Returns le terrain sous forme de chaine de caractères

Return type str

CHAPTER

SIX

STRATEGIE

class Strategie.**Strategie**(*robot*)

Classe abstraite à implémenter pour faire une stratégie

Parameters **robot** (*Robot*) – Le robot devant prendre une decision

decider()

Méthode abstraite, retourne la liste des mouvements à effectuer après analyse du terrain pour la prise de décision

Returns la liste des nouveaux mouvements à effectuer

Return type List<Mouvement>

Raises **NotImplementedError** si la méthode n'a pas été redéfinie.

envoyerUnites(*depuis_cellule*, *vers_cellules*, *nb_unites*)

Permet d'envoyer des unités attaquantes d'une cellule vers une autre, ne fait que modifier le terrain (ne les envoie pas au serveur). On modifie le terrain afin de ne pas reprendre en compte les unitées déjà utilisées. Retourne le mouvement créé !

Parameters

- **depuis_cellule** (*Cellule*) – La cellule qui envoie les unités attaquantes
- **vers_cellules** (*Cellule*) – La cellule qui reçoit les unités attaquantes

Returns le mouvement créé

Return type Mouvement

getRobot()

Retourne le robot

Returns le robot

Return type Robot

CHAPTER

SEVEN

STRATEGIEANALYSE

class StrategieAnalyse.**StrategieAnalyse**(*robot*)

Cette stratégie analyse le terrain afin de déduire où elle doit envoyer les unités des cellules.

L'envoi des unités ainsi que leur destination sera déterminée après analyse du terrain. Les cellules seront réparties en deux groupes : les productrices et les attaquantes

Parameters **robot** (*Robot*) – Le robot devant prendre une decision

decider()

Retourne la liste des mouvements à effectuer après analyse du terrain pour la prise de décision.

Returns la liste des nouveaux mouvements à effectuer

Return type List<Mouvement>

determinerCible(*attaquante*)

Détermine la cible d'une cellule attaquante (en utilisant l'indice P sur tous les voisins ennemies de la cellule attaquante).

Parameters **attaquante** (*Cellule*) – la cellule attaquante cherchant une cible

Returns la cellule cible

Return type Cellule

envoyerUnitesAttaquantes(*composante*, *mesCellules*)

Retourne la liste des mouvements correspondant aux mouvements des unités des cellules attaquantes vers les cellules ennemies les plus prometteuses.

Parameters

- **composante** (*Terrain*) – la partie du terrain (sous graphe) où se trouvent nos cellules
- **mesCellules** (*dict of list of Cellule*) – dictionnaire de liste de cellules

Returns la liste des mouvements des unités des cellules attaquantes vers les cellules ennemies les plus prometteuses.

Return type List<Mouvement>

envoyerUnitesProductrices(*composante*, *mesCellules*)

Retourne la liste des mouvements correspondant aux mouvements des unités des cellules productrices vers les cellules attaquantes les plus proches.

Parameters

- **composante** (*Terrain*) – la partie du terrain (sous graphe) où se trouvent nos cellules
- **mesCellules** (*dict of list of Cellule*) – dictionnaire de liste de cellules

Returns la liste des mouvements des unités des cellules productrices vers les cellules attaquantes les plus proches.

Return type List<Mouvement>

getAttaquantesEnDanger (*attaquantes*)
Retourne la liste de nos cellules attaquantes en danger. Une cellule attaquante est en danger lorsqu'il y a des unités ennemies se dirigeant vers elle.

Parameters **attaquantes** (*List<Cellule>*) – la liste de toutes nos cellules attaquantes

Returns nos cellules attaquantes en danger

Return type List<Cellule>

getAttaquantesEnSurete (*attaquantes*, *attaquantes_en_dangers*)
Retourne la liste de nos cellules attaquantes en sûreté

Return type List<Cellule>

getCellulesAttaquantes (*mesCellules*)
Retourne la liste de nos cellules attaquantes. Une cellule est attaquante si elle est reliée à au moins un ennemi.

Parameters **mesCellules** (*List<Cellule>*) – nos cellules attaquantes

Return type List<Cellule>

getCellulesProductrices (*mesCellules*)
Retourne la liste des cellules productrices. Une cellule est productrice si elle n'est reliée à aucun ennemi.

Parameters **mesCellules** (*List<Cellule>*) – nos cellules productrices

Return type List<Cellule>

getCoutCellule (*cellule*)
Calcule le nombre d'unités que l'on doit envoyer sur une cellule ennemie afin de la capturer Ce coût peut être négatif si la cellule nous appartient déjà.

Parameters **cellule** (*Cellule*) – Cellule ennemie

Returns le nombre d'unités nécéssaire à la capture de la cellule

Return type int

getFullProductrices (*productrices*, *semi_productrices*)
Retourne la liste de nos cellules entièrement productrices (full-productrices). Une cellule est full-productrice si c'est une cellule productrice qui n'est reliée à aucune cellule attaquante.

Return type List<Cellule>

getMesCellules ()
Retourne la liste des cellules nous appartenant

Returns les cellules nous appartenant

Return type List<Cellule>

getSemiProductrices (*productrices*, *attaquantes*)
Retourne la liste de nos cellules semi-productrices. Une cellule est semi-productrice si c'est une cellule productrice reliée à au moins une cellule attaquante

Returns nos cellules semi-productrices

Return type List<Cellule>

indiceP (*origine*, *cellule*)
Calcule l'indice P d'une cellule par rapport à la cellule d'origine voulant envoyer ses unités.

Parameters

- **origine** (*Cellule*) – La cellule d'origine
- **cellule** (*Cellule*) – La cellule dont on veut calculer l'indice P

Returns l'indice P de la cellule cible

Return type float

nbUnitesAEnvoyer (*attaquante*, *cellule_cible*)
Détermine le nombre d'unités à envoyer de la cellule attaquante vers une cellule_cible en fonction de l'excédent de l'attaquant et du coût de la cible. Peut renvoyer 0, dans ce cas la, la cellule attaquante ne devra pas envoyer d'unités.

Parameters

- **attaquante** (*Cellule*) – la cellule de départ voulant déterminer le nombre d'unités à envoyer
- **cellule_cible** (*Cellule*) – la cellule ciblée

Returns le nombre d'unités à envoyer vers la cellule ciblée

Return type int

CHAPTER

EIGHT

STRATEGIEPREVISION

class StrategiePrevision.**StrategiePrevision**(*robot*)

Cette stratégie est une surcouche de la stratégie Analyse. Elle essaie de prendre les cellules au bon moment, afin de subir le moins de pertes possible

Parameters **robot** (*Robot*) – Le robot devant prendre une decision

determinerCible(*attaquante*)

Dértermine la cible d'une cellule attaquante

Parameters **attaquante** (*Cellule*) – la cellule attaquante cherchant une cible

Returns la cellule cible

Return type Cellule

CHAPTER

NINE

STRATEGIEALEATOIRE

class StrategieAleatoire.**StrategieAleatoire**(*robot*)

1ere stratégie qui attaque aléatoirement.

L'envoi des unités ainsi que leur destination sera déterminée aléatoirement.

Parameters **robot** (*Robot*) – Le robot devant prendre une decision

decider()

Retourne la liste des mouvements à effectuer après analyse du terrain pour la prise de décision.

Returns la liste des nouveaux mouvements à effectuer

Return type List<Mouvement>

CHAPTER

TEN

GRAPHIQUE

class Graphique.**Graphique**(*robot*)

Représente L'interface graphique du jeu.

Parameters **robot** (*Robot*) – Le robot du jeu

create_circle(*x*, *y*, *r*, ***kwargs*)

Créer un cercle dans le canvas

Parameters

- **x** (*int*) – La position en X du centre du cercle
- **y** (*int*) – La position en Y du centre du cercle
- **r** (*int*) – Le rayon du cercle
- ****kwargs** – Paramètres facultatifs de tkinter pour créer un cercle

decalage_centre_cercle(*u*, *v*)

Permet de récupérer le point d'intersection entre le bord du cercle et le lien.

Parameters

- **u** (*Cellule*) – La cellule de départ
- **v** (*Cellule*) – La cellule d'arrivée

dessinerCellule(*cellule*)

Permet de dessiner une cellule dans la fenêtre

Parameters **cellule** (*Cellule*) – La cellule à dessiner

dessinerCellules()

Utilise la méthode "dessinerCellule" pour dessiner l'ensemble des cellules du jeu dans la fenêtre

dessinerLien(*lien*)

Permet de dessiner un lien dans la fenêtre.

Parameters **lien** (*Lien*) – Le lien à dessiner

dessinerLiens()

Utilise la méthode "dessinerLien" pour dessiner l'ensemble des liens sur la fenêtre

dessinerMouvement(*mouvement*)

Permet de dessiner un mouvement se déplaçant sur un lien

Parameters **mouvement** (*Mouvement*) – le mouvement à dessiner

dessinerMouvements()

Utilise la méthode "dessinerMouvement" pour dessiner tous les mouvements en cours du terrain

getTrueCoordonneeCellule(*cellule*)
Permet d'adapter les coordonnées de la cellule envoyées par le serveur en fonction des dimensions de la fenêtre

Parameters **cellule** (*Cellule*) – La cellule à adapter

Return type float

getTrueRayonCellule(*cellule*)
Permet d'adapter le rayon de la cellule envoyé par le serveur en fonction des dimensions de la fenêtre

Parameters **cellule** (*Cellule*) – La cellule à adapter

Return type float

redessinerCellule(*cellule*)
Permet de redessiner une cellule

Parameters **cellule** (*Cellule*) – La cellule à redessiner

redessinerCellules()
Utilise la méthode "redessinerCellule" pour redessiner toutes les cellules suivant l'évolution du jeu.

class Graphique.**Point**(*x*, *y*)
Permet de créer un point aux coordonnées spécifiées en paramètre.

class Graphique.**Vecteur**(*a*, *b*)
Permet de représenter un vecteur avec les coordonnées spécifiées en paramètre.

norme()
Calcule la norme du vecteur

Return type float

produitScalaireBAC(*ab*, *ac*)
Retourne le produit scalaire des deux vecteurs passés en paramètre

Parameters

- **ab** (*Vecteur*) – Le premier vecteur
- **ac** (*Vecteur*) – Le second vecteur

Return type float

rotation(*angle*)
Rotation d'un vecteur de l'angle passé en paramètre

Parameters **angle** (*float*) – l'angle de rotation à appliquer au vecteur (en radians)

translationPoint(*point*)
Effectue une translation du point passé en paramètre par rapport au vecteur

Parameters **point** (*Point*) – Le point à translater

Return type Point

CHAPTER

ELEVEN

EXCEPTIONS

exception Exceptions.**CelluleException**(*message*)
Se déclenche lors d'une exception dans la classe Cellule

exception Exceptions.**LienException**
Se déclenche lors d'une exception dans la classe Lien

exception Exceptions.**MouvementException**
Se déclenche lors d'une exception dans la classe Mouvement

exception Exceptions.**RobotException**
Se déclenche lors d'une exception dans la classe Robot

exception Exceptions.**TerrainException**
Se déclenche lors d'une exception dans la classe Terrain

CHAPTER

TWELVE

GRAPHIQUE_LOLLIPOOO

graphique_lolipooo.**register_pooo**(*uid*)
Inscrit un joueur et initialise le robot pour la compétition

Parameters **uid** (*chaîne de caractères str(UUID)*) – identifiant utilisateur

Example

"0947e717-02a1-4d83-9470-a941b6e8ed07"

graphique_lolipooo.**init_pooo**(*init_string*)
Initialise le robot pour un match

Parameters **init_string** (*chaîne de caractères (utf-8 string)*) – instruction du protocole de communication de Pooo (voire ci-dessous)

INIT<matchid>TO<#players>[<me>];<speed>; <#cells>CELLS:<cellid>(<x>,<y>)'<radius>'<offsize>'<defsize>'<prod>,...; <#lines>LINES:<cellid>@<dist>OF<cellid>,...

<me> et <owner> désignent des numéros de 'couleur' attribués aux joueurs. La couleur 0 est le neutre. le neutre n'est pas compté dans l'effectif de joueurs (<#players>). '...' signifie que l'on répète la séquence précédente autant de fois qu'il y a de cellules (ou d'arêtes). 0CELLS ou 0LINES sont des cas particuliers sans suffixe. <dist> est la distance qui sépare 2 cellules, exprimée en... millisecondes ! /!attention: un match à vitesse x2 réduit de moitié le temps effectif de trajet d'une cellule à l'autre par rapport à l'indication <dist>. De manière générale temps_de_trajet=<dist>/vitesse (division entière).

Example

"INIT20ac18ab-6d18-450e-94af-bee53fdc8fcaTO6[2];1;3CELLS:1(23,9)'2'30'8'I,2(41,55)'1'30'8'II,3(23,103)'1'20'5'I;2LINE

graphique_lolipooo.**play_pooo**()
Active le robot-joueur

graphique_lolipooo.**updateGraphique**(*graphique*)
Permet la mise à jour automatique de l'interface graphique

Parameters **graphique** (*Graphique*) – l'interface graphique

graphique_lolipooo.**updateTime**(*robot*)
Permet la mise à jour automatique du temps.

Parameters **robot** (*Robot*) – le robot

graphique_lolipooo.**updateGame**(*robot*)
Permet la mise à jour automatique de l'état du jeu (terrain...).

Parameters **robot** (*Robot*) – le robot

graphique_lolipooo.**sendDecisions**(*robot*)
Permet l'envoi automatique des decisions au serveur.

Parameters **robot** (*Robot*) – le robot

CHAPTER

THIRTEEN

LOLLIPOOO

lolipooo.**register_pooo**(*uid*)
Inscrit un joueur et initialise le robot pour la compétition

Parameters **uid** (*chaîne de caractères str(UUID)*) – identifiant utilisateur

Example

"0947e717-02a1-4d83-9470-a941b6e8ed07"

lolipooo.**init_pooo**(*init_string*)
Initialise le robot pour un match

param init_string instruction du protocole de communication de Pooo (voire ci-dessous)

type init_string chaîne de caractères (utf-8 string)

INIT<matchid>TO<#players>[<me>];<speed>; <#cells>CELLS:<cellid>(<x>,<y>)'<radius>'<offsize>'<defsize>'<prod>,...; <#lines>LINES:<cellid>@<dist>OF<cellid>,...

<me> et <owner> désignent des numéros de 'couleur' attribués aux joueurs. La couleur 0 est le neutre. le neutre n'est pas compté dans l'effectif de joueurs (<#players>). '...' signifie que l'on répète la séquence précédente autant de fois qu'il y a de cellules (ou d'arêtes). 0CELLS ou 0LINES sont des cas particuliers sans suffixe. <dist> est la distance qui sépare 2 cellules, exprimée en... millisecondes ! /!attention: un match à vitesse x2 réduit de moitié le temps effectif de trajet d'une cellule à l'autre par rapport à l'indication <dist>. De manière générale temps_de_trajet=<dist>/vitesse (division entière).

Example

"INIT20ac18ab-6d18-450e-94af-bee53fdc8fcaTO6[2];1;3CELLS:1(23,9)'2'30'8'I,2(41,55)'1'30'8'II,3(23,103)'1'20'5'I;2

lolipooo.**play_pooo**()
Active le robot-joueur

lolipooo.**updateGraphique**(*graphique*)
Permet la mise à jour automatique de l'interface graphique

Parameters **graphique** (*Graphique*) – l'interface graphique

lolipooo.**updateTime**(*robot*)
Permet la mise à jour automatique du temps.

Parameters **robot** (*Robot*) – le robot

lolipooo.**updateGame**(*robot*)
Permet la mise à jour automatique de l'état du jeu (terrain...).

Parameters **robot** (*Robot*) – le robot

lolipooo.**sendDecisions**(*robot*)
Permet l'envoi automatique des decisions au serveur.

Parameters **robot** (*Robot*) – le robot

c

e

g

l

m

r

s

t