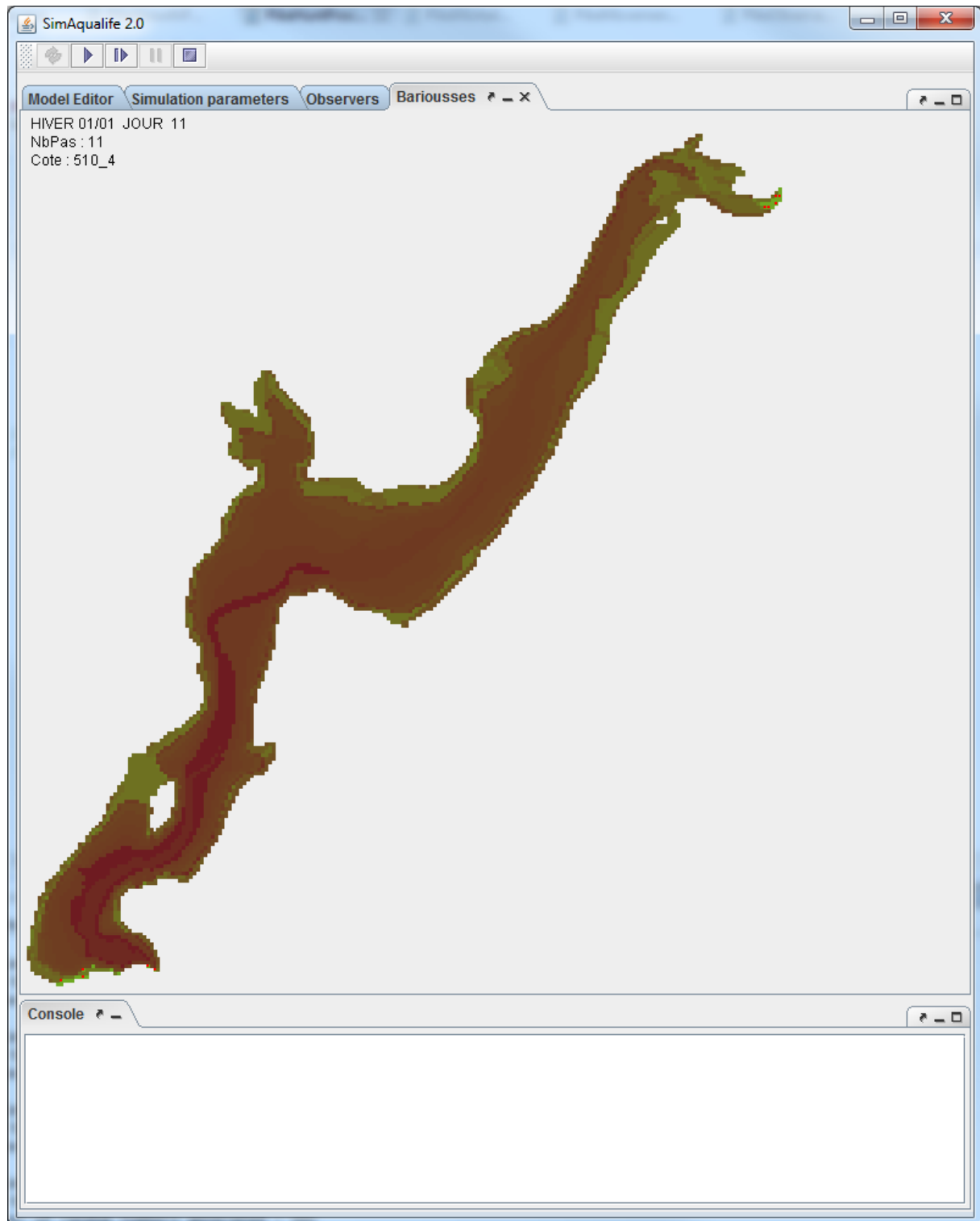


PikeLake



Sommaire

I.	Description	4
II.	Individus	4
III.	Environnement	4
IV.	Process.....	5
A.	Process à l'Initialisation	5
1.	HorairesLeverCoucher	5
2.	FichierMarnage.....	5
3.	AreaMovement	5
4.	PikesPopulateProcess.....	6
B.	Process à Chaque pas de temps	6
1.	Time	6
2.	Marnage	6
3.	PikeMovement	7
4.	PikeTrackLocation.....	7
C.	Process à la Fin de la Simulation	7
1.	SaveLocation.....	7
D.	Process à créer	7
V.	Fichiers d'Entrée et de Sortie	8
A.	Entrée	8
B.	Sortie	9
VI.	Lancement de la simulation	9
VII.	Package pikelake :	10
A.	Cell.java	10
B.	Grid.java	11
C.	Individual.java.....	12
D.	Marnage.java.....	12
E.	SaveLocation.java	13
VIII.	Package pikelake.environment:	14
A.	AreaMovement.java	14
B.	FichierMarnage.java	15
C.	HorairesLeverCoucher.java	15
D.	Movement2DWithinShapeObserver.java	16

E.	Time.java	17
IX.	Package pikelake.pikes:	19
A.	Pike.java.....	19
B.	PikeGrowthProcess.java	20
C.	PikeHuntProcess.java	20
D.	PikeMortalityProcess.java	21
E.	PikeMovement.java.....	21
F.	PikeObservationProcess.java	21
G.	PikePlopProcess.java	22
H.	PikeReproductionProcess.java	22
I.	PikesGroup.java.....	22
J.	PikesPopulateProcess.java	23
K.	PikeTrackLocation.java	24

I. Description

Programme de modélisation des déplacements de brochets dans un lac : le lac des Bariousses. Ce modèle dépend du projet SimAquaLife.

Ce projet se base sur le projet PredatorPrey, qui simule les déplacements, la croissance, la reproduction ainsi que le décès de deux populations distinctes : les prédateurs et les proies. Certains process et certaine classes sont héritées de ce projet mais ne sont pas utilisés dans le projet PikeLake (PikeGrowthProcess, PikeHuntProcess, PikeMortalityProcess, PikeReproductionProcess...).

Le modèle commence par placer aléatoirement le nombre d'individus (défini par l'utilisateur) dans le lac. La simulation débute le 01/01, chaque pas de temps correspond à une heure, la durée de la simulation à calculer par l'utilisateur est donc en nombre de pas de temps (ex : 1 an = 24h * 365j = 8760 pas de temps).

A chaque pas de temps (horaire dans ce programme), les individus se déplacent dans un rayon d'action défini par saison et phase du jour. Au sein de ce rayon, ils font un choix de destination selon le « Habitat Suitability Index (HSI) » des mailles du lac fonction de la saison et de la cote du plan d'eau. Ce HSI a été calculé à partir des préférences d'habitat étudiées avec les ratios de sélection.

D'un point de vue calendrier, on a choisi une année type qui se répète, il s'agit de l'année 2012, avec ses caractéristiques de marnage, ses horaires de lever et de coucher du soleil.

II. Individus

Ce modèle permet de simuler les déplacements d'une population de brochets.

III. Environnement

Le lac qui a été modélisé avec ce programme est le lac des Bariousses situé sur la Vézère à Treignac en Corrèze. Ce lac a été créé avec la construction d'un barrage, de ce fait le niveau de l'eau varie. Le marnage de ce lac (hors vidange) se situe entre 507m et 513,5m.

Pour le modèle, le lac a été discrétisé en maille de 10m*10m. On attribue ensuite à chaque maille une valeur de HSI (Habitat Suitability Index), cette valeur permet de déterminer l'attractivité de chaque maille pour les brochets.

Ces valeurs ont été déterminées en fonction du type de substrat, du type de végétation (pour les rives), de la profondeur de chaque maille et ce pour chaque saison. Comme le niveau du lac n'étant pas constant ces calculs ont dû être effectués pour chaque cote située entre 507m et 513,5m (avec un pas de 0,1m).

Des fichiers textes ont été créés avec ces valeurs, ils sont dans le modèle dans le dossier data/input/HSI. Chaque fichier est associé à un marnage et une saison. Ces fichiers contiennent autant de ligne qu'il y a de cellule dans le lac pour un marnage. Chaque ligne contient l'id d'une cellule et le HSI qui lui est associé.

IV. Process

Les process permettent d'effectuer divers traitements que ce soit sur un individu ou toute la population mais aussi sur des variables définissant l'environnement, la gestion du temps...

Ces traitements peuvent être effectués une seule fois au début ou à la fin, mais aussi à chaque pas de la simulation.

Un process doit être défini dans une classe dans la méthode `doProcess()`. Cette classe doit être spécifiée dans le fichier *fishLight.xml*, il est également possible d'initialiser les variables de ces classes dans ce fichier.

A. Process à l'Initialisation

1. HoraireLeverCoucher

Process qui permet de lire les informations du fichier *leshorairesdusoleil.csv*. Ce fichier contient les horaires de lever et du coucher du soleil pour une année type (2012).

Ce process range ensuite ces horaires dans le tableau *heureLeverCoucher* [0=lever,1=coucher] [jour] [mois]. Ces horaires permettent ensuite de remplir le tableau *phaseJournée* [jour] [mois] [heure], ce dernier renvoie une chaîne de caractère indiquant la phase de la journée pour une date donnée (mois + jour + heure).

Une Journée est découpée en quatre phases :

- Aube : +1h et -1h par rapport à l'heure de lever du soleil (durée = 3h)
- Jour
- Crépuscule : +1h et -1h par rapport à l'heure de coucher du soleil (durée = 3h)
- Nuit

2. FichierMarnage

Process qui lit le contenu du fichier *CoteCorrige.txt*. Ce fichier donne la cote du lac en fonction de la date (basée sur une année type : 2012). Il range ensuite, dans le tableau *dateMarnage* [jour] [mois] [heure], la valeur de chaque côte pour chaque date (mois, jour, heure).

3. AreaMovement

Process qui lit le fichier contenant les caractéristiques (max, min, moyenne et écart type) des distances horaires parcourues par les individus, selon la saison et la phase du jour. Trois distances sont disponibles (min, moy, max), le choix de la distance utilisée est effectué dans le fichier fishLight.xml

Une telle distance correspond à un rayon d'action d'un individu qui va permettre de définir l'ensemble des mailles qui lui sont potentiellement accessibles en un pas de temps (horaire ici). Ici, ce rayon d'action ne dépend que de saison et phase du jour (soit $4 \times 4 = 16$ possibilités). Pour chaque rayon, on a donc calculé les coordonnées (ou identifiants) des mailles accessibles pour un individu qui serait au point (0,0).

Dans le cours du programme, il suffira alors d'ajouter ces coordonnées à celles de la maille où se trouve un individu pour obtenir l'ensemble des mailles qui lui sont accessibles, en ne conservant que celles qui sont bel et bien dans le lac. Remplit les tableaux contenant les coordonnées des cellules accessibles. Cet ensemble de coordonnées représente un cercle de rayon égal à la distance de déplacement.

Afin d'obtenir les cellules accessibles pour un individu il faut ajouter chacune de ces coordonnées à la position de l'individu à déplacer et vérifier que cette nouvelle coordonnée se trouve dans le lac.

4. PikesPopulateProcess

Process qui permet d'initialiser l'ensemble de la population de la simulation. Il crée le nombre d'individus voulus pour la simulation et les places aléatoirement sur le lac.

Il n'est pas possible (avec le modèle actuel) de définir les coordonnées initiales pour chaque individu.

B. Process à Chaque pas de temps

1. Time

Process qui gère le temps lors de la simulation. Lors de l'exécution de la simulation chaque pas de temps correspond à une heure. Ce process permet, à partir du nombre de pas de temps de la simulation de calculer l'ensemble des variables qui sont utiles pour le déroulement de la simulation sur une année.

Il calcule l'heure, le jour, le mois, et l'année de la simulation, mais également la saison ainsi que la phase du jour.

2. Marnage

Process qui permet de modifier l'étendue du lac suivant le marnage. Il met à jour le nombre de cellules accessibles dans le lac, ainsi que le HSI de chacune d'entre elles.

Pour ne pas charger inutilement à chaque pas de temps la grille contenant le lac (environ 8 000 cellules selon la cote), ce process n'est effectué que si la valeur du marnage est modifiée d'une heure à l'autre.

3. PikeMovement

Process qui permet d'effectuer les déplacements des individus à l'intérieur du lac. La liste des cellules accessibles est calculée à partir de la liste de cellule donnée par le process AreaMovement auquel il faut ajouter la position de l'individu afin de ne garder que celles comprises dans le lac. Ce process choisit la cellule avec le HSI le plus élevé parmi la liste des cellules accessibles.

4. PikeTrackLocation

Process qui enregistre dans un tableau la position de chaque individu, ainsi que la date correspondante.

C. Process à la Fin de la Simulation

1. SaveLocation

Process qui sauvegarde la position des individus à la fin de la simulation. Les positions (correspondant à l'identifiant de la maille) ainsi que la date correspondant à chacune sont enregistrées dans un fichier de sortie.

Extrait du fichier de sortie :

PasTps	Date	Ind 1	Ind 2	Ind 3	Ind 4	Ind 5	Ind 6	Ind 7	...
1	2012/01/01/01	4530	5391	5606	5392	52066	52282	51444	...
2	2012/01/01/02	4962	5389	5178	5607	51432	51211	52067	...
3	2012/01/01/03	5393	4962	4530	5393	51856	51856	51854	...

D. Process à créer

Un process de calcul des distances horaires parcourues

Un process de calcul de l'angle du dernier tronçon de trajectoire par rapport à un axe ouest-est.

V. Fichiers d'Entrée et de Sortie

A. Entrée

Fichiers présents dans le dossier data/input.

CoteCorrige.txt

Fichier qui contient le marnage du lac (en m) pour chaque date, à l'heure près, entre le 01-01-2012 et le 30-04-2014. Le modèle se base sur l'année 2012, seule la première partie du fichier est utilisée. Si la simulation dure plusieurs années, c'est l'année 2012 qui sera répétée. Ce fichier est lu à l'initialisation par la classe FichierMarnage et chaque cote est alors rangée dans un tableau (dateMarnage [jour] [mois] [heure]).

DistanceHoraireCartesienne.txt

Fichier contenant le rayon de déplacement des brochets selon la saison et la phase du jour. Plusieurs valeurs sont disponibles : le rayon de déplacement minimum, moyen, maximum ainsi que l'écart type. Le choix du rayon à utiliser est fait dans le fichier fishLight.xml. Ce fichier est lu par la classe AreaMovement.

Leshorairesdusoleil.csv

Fichier contenant les horaires de lever et de coucher du soleil pour l'année 2012 pour la ville de Brive-la-Gaillarde. Ces données sont lues par la classe HoraireLeverCoucher afin de déterminer chaque phase de la journée.

Dossier HSI

Dossier contenant l'ensemble des fichiers de HSI. Pour chaque marnage et à chaque saison correspond un fichier .txt. Chaque fichier contient l'id de toutes les cellules disponible avec le marnage et la valeur de leur HSI associé.

fishLight.xml

Fichier qui déclare l'ensemble des process utilisés pendant l'exécution du modèle. Ces process peuvent être appelés à l'initialisation, à chaque pas de temps, ou alors à la fin du programme.

Il est également possible de modifier la valeur des variables des classes définissant ces process.

grid.xml

Fichier qui permet de définir l'environnement de la simulation. Largeur et longueur de la grille notamment.

observersCharts.xml
observersBatch.xml

Ces fichiers permettent de définir les observateurs et de modifier ou d'initialiser les variables de ceux-ci.

B. Sortie

Fichier présent dans le dossier data/output

Position.txt

Fichier contenant l'ensemble des positions de chaque individu avec la date associé à chacune. Il est de la forme :

PasTps	Date	Ind 1	Ind 2	Ind 3	Ind 4	Ind 5	Ind 6	Ind 7	...
1	2012/01/01/01	4530	5391	5606	5392	52066	52282	51444	...
2	2012/01/01/02	4962	5389	5178	5607	51432	51211	52067	...
3	2012/01/01/03	5393	4962	4530	5393	51856	51856	51854	...

VI. Lancement de la simulation

La ligne de commande permettant de lancer la simulation est :

```
-d -simDuration 8760 -observers data/input/observersCharts.xml -groups  
data/input/fishLight.xml -env data/input/grid.xml
```

Description des options :

- d : mode debug (affiche message erreurs, avertissements...)
- simDuration : nombre de pas de temps de la simulation (8760 = 1an = 24h*365j)
pour un pas horaire)
- observers : fichier d'initialisation des observateurs
- groups : fichier d'initialisation des groupes et process
- env : fichier d'initialisation de l'environnement

Description des classes

VII. Package pikelake :

A. Cell.java

Classe qui définit chaque cellule du lac. Dans le modèle PikeLake, le lac des Bariousses a été discrétisé en maille de 10m*10m. Chaque maille est représentée par le programme par une instanciation de cette classe.

Variables

private double habitatQuality;

Nombre qui contient le HSI (Habitat Suitability Index) de la cellule.

private transient List<Pike> pikes;

Liste qui contient les individus qui sont présents dans cette cellule.

Méthodes

public Cell (int index, double habitatQuality)

Constructeur de la classe Cell. Le paramètre index représente le numéro de la cellule et habitatQuality est le HSI qui sera affecté à cette cellule.

public List<Pike> getPikes ()

Méthode qui renvoie la liste des individus présents dans la cellule.

public boolean addPike (Pike pike)

Méthode qui permet d'ajouter un individu.

public boolean removePike (Pike pike)

Méthode qui permet d'enlever un individu.

public double getHabitatQuality ()

Méthode qui renvoie la valeur du HSI de la cellule.

B. Grid.java

Classe qui déclare et initialise la grille de la simulation (le lac et cellule non accessible).

Variable

private String marnageInit

Marnage initial, nécessaire car au début de la simulation, lors de la création de la grille, le marnage calculé par la classe FichierMarnage n'est pas disponible.

Méthodes

public Grid (int gridWidth, int gridHeight, NeighborsType neighborsType)

Constructeur de classe. Les paramètres d'entrées représentent respectivement la largeur et la longueur de la grille et le type de voisinage qui sera utilisé.

public void initTransientParameters (Pilot pilot)

Méthode qui déclare la grille totale, toutes les cellules sont initialisées avec un HSI= -1. Ensuite cette méthode lit le fichier .txt contenant le HSI du lac pour le marnage et la saison initiale, puis modifie les cellules qui seront accessibles (à l'intérieur du lac) avec le HSI lu dans le fichier.

public void setCell (int indexCell, double hsiCell)

Méthode qui permet de modifier la valeur du HSI d'une cellule, la cellule à modifier est donnée par indexCell et la nouvelle valeur du HSI est donnée par hsiCell.

public Grid getGrid ()

Méthode qui renvoie un objet qui représente la grille actuelle.

public void addAquaNism (Individual ind, AquaNismsGroup<?, ?> group)

Méthode qui permet d'ajouter un individu (ind) à un groupe (group).

public void moveAquaNism (Individual ind, AquaNismsGroup<?, ?> group, Cell dest)

Méthode qui permet de déplacer un individu (ind) qui appartient à un groupe (group) dans une cellule donnée (dest).

public void removeAquaNism (Individual ind, AquaNismsGroup<?, ?> group)

Méthode qui permet d'enlever un individu (ind) à un groupe (group).

public List<Cell> getNeighbours (Cell position)

Méthode qui détermine l'ensemble des cellules accessibles (comprise dans le lac). Elle récupère l'ensemble des coordonnées disponible selon le rayon d'action (tableaux donné par la classe AreaMovement).

Ensuite, pour chaque individu, elle ajoute chacune de ces coordonnées à la position de l'individu, si les coordonnées calculées sont comprises dans le lac, on ajoute cette cellule à la liste des cellules accessibles.

Le choix de la cellule de destination se fait ultérieurement.

C. Individual.java

Classe qui permet de définir un individu et toutes les caractéristiques qui le définissent.

Variables

protected int age;

Nombre entier représentant l'âge de l'individu.

protected double weight;

Nombre qui définit le poids de l'individu.

protected boolean sexe;

Bit qui définit le sexe de l'individu. (0: Femelle, 1: Male)

Méthodes

public Individual (Pilot pilot, Cell position, double weight)

Constructeur de classe. Permet de créer un individu, définissant la cellule de destination et le poids de celui-ci.

public final double getWeight ()

Méthode qui renvoie le poids de l'individu.

public final void setWeight (double weight)

Méthode qui permet de définir le poids de l'individu passé en paramètre (weight).

public final int getAge ()

Méthode qui renvoie l'âge de l'individu.

public final void incAge ()

Méthode qui permet d'incrémenter l'âge de l'individu.

public int compareTo (Individual ind)

Méthode qui compare le poids de l'individu actuel avec un autre individu (ind).

D. Marnage.java

Classe qui met à jour les cellules accessibles du lac, ainsi que leurs HSI, en fonction du marnage horaire donné par la classe FichierMarnage.

Variable

public static String marnageOld = null, marnageNew = null;

Elles servent à déterminer, à chaque pas de temps, si le marnage est modifié.

Méthodes

public void doProcess (PikesGroup object)

Teste si la cote du lac est modifiée, si oui MAJ de la grille (appel à la fonction majCote).

public void majCote (Grid grid)

MAJ de la grille, augmentation ou diminution du nombre de cellules accessibles et modification du HSI de chaque cellule du lac.

E. SaveLocation.java

Classe qui sauvegarde la position des individus à la fin de la simulation. Les positions (correspondant à l'identifiant de la maille) ainsi que la date correspondant à chacune sont enregistrées dans un fichier de sortie.

Extrait du fichier de sortie :

PasTps	Date	Ind 1	Ind 2	Ind 3	Ind 4	Ind 5	Ind 6	Ind 7	...
1	2012/01/01/01	4530	5391	5606	5392	52066	52282	51444	...
2	2012/01/01/02	4962	5389	5178	5607	51432	51211	52067	...
3	2012/01/01/03	5393	4962	4530	5393	51856	51856	51854	...

Méthode

public void doProcess (PikesGroup group)

Procédure appelée à la fin de la simulation, elle sauvegarde la position des individus (identifiant de la maille) dans un fichier de sortie. Elle fait appel aux tableaux créés dans la classe PikeTrackLocation, qui contiennent la date et les positions correspondantes pour chaque individus, afin de créer le fichier de sortie au format .txt.

VIII. Package pikelake.environment:

A. AreaMovement.java

Classe qui lit le fichier d'entrée contenant les caractéristiques des distances horaires parcourues par les individus, selon la saison et la phase du jour.

Trois distances sont disponibles (min, moy, max), le choix de la distance utilisée est fait dans le fichier fishLight.xml

Variables

```
private int distMin, distMoy, distMax, std, choixDist;  
private String saison = null, phaseJour = null;
```

Variables utilisées en interne (de la classe) pour remplir les tableaux contenant les coordonnées des cellules à l'intérieur du rayon d'action.

```
public static int[][] areaPrinAube, areaPrinJour, areaPrinNuit, areaPrinCrep;  
public static int[][] areaEteAube, areaEteJour, areaEteNuit, areaEteCrep;  
public static int[][] areaAutAube, areaAutJour, areaAutNuit, areaAutCrep;  
public static int[][] areaHivAube, areaHivJour, areaHivNuit, areaHivCrep;
```

Tableaux d'entier contenant l'ensemble des coordonnées des cellules accessibles qui définissent le cercle des cellules accessibles.

Méthodes

```
public void doProcess (PikesGroup object)
```

Lit le fichier contenant les distances horaires en fonction de la saison et de la phase du jour.
Remplit les tableaux contenant les coordonnées des cellules accessibles.

```
public int [][] choixDistArea ()
```

Détermine la distance qui sera utilisée pour calculer les coordonnées des cellules du cercle de déplacement.

```
public int [][] calculArea (int distance)
```

Calcule les coordonnées des cellules accessibles pour une distance passée en paramètre d'entrée. L'ensemble de ces coordonnées forment un cercle de rayon = distance.

```
public int getDistMin ()
```

Retourne la distance minimum.

```
public int getDistMoy ()
```

Retourne la distance moyenne.

```
public int getDistMax ()
```

Retourne la distance maximum.

```
public int getDistStd ()
```

Retourne l'écart type de la distance (std).

```
public String getSaison ()
```

Retourne la saison.

```
public String getPhaseJour ()
```

Retourne la phase du jour.

B. FichierMarnage.java

Classe qui lit le fichier contenant les différents marnages du lac à chaque date (mois, jour, heure), ces valeurs sont rangées à l'initialisation dans un tableau pour un accès plus rapide.

Variable

```
public static String dateMarnage [jour] [mois] [heure]
```

Tableau comportant la valeur du marnage pour chaque date de la simulation.

Ces valeurs sont des chaîne de caractères, Le « . » séparant les unités des dixièmes est remplacé par un « _ » pour correspondre aux noms des fichiers texte comportant les HSI de chaque cellule pour chaque valeur de la cote (ex : côte 512.4 ➔ 512_4).

Méthodes

```
public void doProcess (PikesGroup object)
```

Lis le fichier contenant les marnages pour le lac et complète le tableau dateMarnage.

```
public static String calculMarnage (int jour, int mois, int heure)
```

Retourne le marnage du lac en fonction d'une date (jour + mois + heure)

C. HoraireLeverCoucher.java

Classe qui lit le fichier contenant

les horaires de lever et de coucher du soleil, ces horaires sont rangés à l'initialisation dans un tableau pour un accès plus rapide.

Variables

```
heureLeverCoucher [0=lever, 1=coucher] [jour] [mois]
```

Tableau d'entiers qui renvoie l'heure de lever ou de coucher du soleil pour chaque date.

```
phaseJournee [jour] [mois] [heure]
```

Tableau de chaîne de caractères qui renvoie la phase du jour en fonction de la date.

Méthodes

`public void doProcess (PikesGroup object)`

Lis le fichier d'entrée contenant les horaires de lever et de coucher du soleil, puis remplis les tableaux `heureLeverCoucher` et `phaseJournee`.

`public static String getPhase (int jour, int mois, int heure)`

Renvoie la phase du jour en chaîne de caractère ("AUBE", "JOUR", "CREP", "NUIT"), selon la date donnée par les paramètres en entrées : jour, mois, heure.

`public int getHoraire (int levercoucher, int jour, int mois)`

Retourne l'heure de lever ou de coucher du soleil en fonction d'une date (jour + mois). Le paramètre `levercoucher` permet de choisir si l'on veut l'heure de lever du soleil (`levercoucher=0`) ou alors celle du coucher (`levercoucher=1`).

D. Movement2DWithinShapeObserver.java

Classe qui permet d'afficher le lac ainsi que les individus à chaque pas de temps lors de l'exécution du programme.

Variables

`private String title;`

Titre de la fenêtre qui affiche le lac.

`private Color bgColor;`

Couleur de l'arrière-plan.

`private Color pikeColor;`

Couleur des brochets.

`private Color textColor;`

Couleur du texte des annotations.

`private double hsiMaxValueThreshold=0.5;`

Valeur maximum du HSI pour l'affichage dégradé.

Méthodes

`public Movement2DWithinShapeObserver (int margin, Color bgColor, Color shapeColor)`

Constructeur de classe.

`public String getTitle ()`

Renvoie le titre de la fenêtre.

public void init (Pilot pilot)
Méthode qui initialise l’affichage du lac.

protected synchronized void paintComponent (Graphics g)
Méthode qui permet de dessiner le lac ainsi que les individus.

E. Time.java

Classe de la gestion du temps pendant la simulation.

Variables

public static enum Season {PRINTEMPS, ETE, AUTOMNE, HIVER};
Énumération comportant les saisons.

public static enum Mois {MoisZero, JANVIER, FEVRIER, MARS, AVRIL, MAI, JUIN,
JUILLET, AOÛT, SEPTEMBRE, OCTOBRE, NOVEMBRE, DECEMBRE};
Énumération comportant les mois. Le MoisZero a été ajouté pour faire correspondre le numéro du mois à celui de tout calendrier :

- JANVIER est le mois 1 et pas 0.
- DECEMBRE est le mois 12 et pas 11.

public static int mois, jour, jourMois, saison, année, phaseJour, heure;
Permettent d’obtenir une date complète. Attention à ne pas confondre les variables jour et jourMois :

- Jour représente le nombre de jours dans l’année (1 – 365).
- JourMois représente le nombre de jour dans le mois (1 – 30,31).

public static String PhaseJour;
Représente la phase du jour de la simulation.

public static long nbrIter;
Représente le nombre de pas de temps écoulé depuis le début de la simulation (= nombre d’heures).

Méthodes

public void doProcess (PikesGroup group)
Détermine la date de la simulation en fonction du nombre de pas de la simulation, chaque pas de temps de la simulation est équivalent à une heure.
Calcule l’heure, le jour, la phase du jour, le mois, la saison et l’année.

public void calculMois ()
Détermine le mois en fonction du jour par an. (1 <= jour par an <= 365)

```
public void calculSaison ()
```

Détermine la saison en fonction du mois et du jour de la simulation.

```
public static String getPhaseJour ()
```

Renvoie une chaîne de caractères représentant la phase du jour en cours (AUBE, JOUR, CREP, NUIT).

```
public static String getMois ()
```

Renvoie une chaîne de caractères représentant le mois en cours (JANVIER, FEVRIER, MARS, AVRIL, MAI, JUIN, JUILLET, AOUT, SEPTEMBRE, OCTOBRE, NOVEMBRE, DECEMBRE)

```
public static String getSeason ()
```

Renvoie une chaîne de caractères représentant la saison en cours (PRINTEMPS, ETE, AUTOMNE, HIVER).

IX. Package pikelake.pikes:

A. Pike.java

Classe qui définit chaque brochet avec ses caractéristiques. Hérite de la classe Individual.

Variables

`private double ingestedFood;`

Variable qui représente la nourriture qui est ingérée par le brochet.

`public static int cptIndividu;`

Variable qui est incrémentée à la création de chaque individu. Après l'initialisation elle contient le nombre total d'individu.

`public int idIndividu;`

Variable qui permet de différencier chaque brochet. A chaque création d'un brochet elle est incrémentée.

Méthodes

`public Pike (Pilot pilot, Cell cell, int age, double weight)`

Constructeur de la classe Pike. Les paramètres représentent la cellule dans laquelle sera créé le brochet, l'âge de celui-ci et son poids.

`public final int getIdIndividu ()`

Méthode qui renvoie l'id du brochet. Lors de la création chaque brochet est numéroté pour pouvoir les différencier.

`public double getIngestedFood ()`

Méthode qui renvoie la valeur qui représente la nourriture ingérée par le brochet.

`public void setIngestedFood (double ingestedFood)`

Méthode qui permet de définir la valeur qui représente la nourriture ingérée par le brochet.

`public void addIngestedFood (double ingestedFood)`

Méthode qui permet de d'ajouter la valeur qui représente la nourriture ingérée par le brochet à celle qu'il a déjà ingéré.

`public double getSuitabilityForPike (Cell cell)`

Méthode qui renvoie le HSI de la cellule passé en paramètre (cell), si un autre brochet est présent dans cette cellule cette méthode renvoie -1.

B. **PikeGrowthProcess.java**

Classe qui gère la croissance des individus.

Variable

```
private double conversionFactor;
```

Variable qui représente le pourcentage de nourriture ingéré par l'individu qui sera ajouté à son poids.

Méthode

```
public void doProcess (PikesGroup group)
```

Process qui permet de gérer la croissance des individus. Permet d'augmenter le poids de ceux-ci lorsqu'ils ingèrent une proie.

C. **PikeHuntProcess.java**

Classe qui permet aux individus de chasser des proies.

Variables

```
private double satiety = 0.10;
```

Ratio qui détermine lorsque l'individu arrête de manger.

```
transient private UniformGen uniformGen;
```

Nombre aléatoire.

Méthodes

```
public void initTransientParameters (Pilot pilot)
```

Méthode d'initialisation de la classe PikeHuntProcess.

```
protected void doProcess (Pike pike, AquaNismsGroup<Pike, ?> group)
```

Méthode permettant aux individus de manger une proie et enlève la proie qui a été mangée, mais ajoute aussi une partie du poids de la proie au prédateur.

D. PikeMortalityProcess.java

Classe qui gère la mort des individus. Si le poids et l'âge de l'individu sont trop élevé et qu'il dépasse le seuil, l'individu est alors déclaré mort et il est enlevé du modèle.

Méthode

```
public void doProcess (PikesGroup group)
```

Méthode qui teste si l'individu a dépassé le seuil, si oui il est alors supprimé de la simulation.

E. PikeMovement.java

Classe qui permet aux individus de se déplacer à l'intérieur du lac. Elle détermine le nombre de cellules accessibles selon le rayon de déplacement, sa position et le marnage du lac. L'individu choisit la cellule ayant le HSI le plus élevé.

Variable

```
transient private UniformGen uniformGen;
```

Nombre aléatoire.

Méthodes

```
public PikeMovement (Pilot pilot)
```

Constructeur de la classe PikeMovement.

```
public void initTransientParameters (Pilot pilot)
```

Méthode qui initialise la variable uniformGen.

```
protected void doProcess (Pike pike, PikesGroup group)
```

Procédure qui récupère la liste des cellules accessibles et qui détermine celle qui a le HSI le plus élevé. L'individu est alors déplacé dans la cellule ayant le HSI le plus élevé.

F. PikeObservationProcess.java

Classe qui déclare les observateurs.

Méthode

```
public void doProcess (PikesGroup pikesGroup)
```

Méthode qui permet de déclarer les observateurs.

G. PikePlopProcess.java

Classe qui permet de mettre en place une temporisation.

Variable

private int temporisation;

Variable représentant la durée de la temporisation en ms.

Méthode

public void doProcess (PikesGroup object)

Méthode qui effectue la temporisation.

H. PikeReproductionProcess.java

Classe qui permet aux individus de se reproduire.

Variables

private int ageAtFirstReproduction;

Représente l'âge à partir duquel l'individu peut se reproduire.

private double pikeFertility = 1.1;

Représente le nombre de descendant par individus.

transient private PoissonGen poissonGen;

Nombre aléatoire.

Méthodes

public void initTransientParameters (Pilot pilot)

Méthode qui initialise le nombre poissonGen.

public void doProcess (PikesGroup group)

Méthode qui effectue la reproduction entre les individus. Elle détermine le nombre de descendant et tue les géniteurs.

I. PikesGroup.java

Classe qui permet de définir des caractéristiques et des comportements généraux pour tous les individus présents dans le modèle.

Variables

private double weightAtAgeThreshold;
Seuil du rapport entre le poids (en g) et l'âge (en année).

private int monthOfBirth;
Mois de la reproduction.

private transient double pikesBiomass;
Somme de tous les poids de chaque individu présent dans le modèle.

private transient int pikesNumber;
Nombre d'individus présent dans tout le modèle.

Méthodes

public PikesGroup (Pilot pilot, Grid arg0, Processes arg1)
Constructeur de la classe.

public int calculatePikesNumber ()
Méthode qui retourne le nombre total d'individus présent dans le modèle.

public double calculatePikesBiomass ()
Méthode qui permet de calculer la biomasse, c'est-à-dire la somme des poids de chaque individu.

public int getMonthOfBirth ()
Méthode qui renvoie le mois de reproduction.

public double getWeightAtAgeThreshold ()
Méthode qui renvoie le seuil weightAtAgeThreshold.

J. PikesPopulateProcess.java

Classe qui permet d'initialiser l'ensemble la population. Elle créer le nombre d'individus spécifié par l'utilisateur et les places aléatoirement sur le lac.

Variables

private int initialNumberOfPikes =5 ;
Nombre initial d'individus. La valeur par défaut est 5, cette valeur est à spécifier dans le fichier fishLight.xml.

transient private UniformGen uniformGen;
Nombre aléatoire utilisé pour placer les individus aléatoirement sur la grille.

Méthodes

public void initTransientParameters (Pilot pilot)

Méthode qui initialise la variable uniformGen.

public void doProcess (PikesGroup group)

Méthode qui est appelée à l'initialisation du programme. Elle déclare le nombre d'individus spécifié par l'utilisateur et les places aléatoirement sur le lac.

K. PikeTrackLocation.java

Classe qui enregistre la position (l'identifiant de la maille) de chaque individu à chaque pas de temps, ainsi que la date et le numéro de pas de temps correspondant. Toutes ces données sont stockées dans des tableaux qui seront lus à la fin de la simulation pour construire le fichier de sorties des positions.

Variables

public static int trackLocation [50] [8760]

Tableau d'entier contenant la position (l'identifiant de la maille) de chaque individu à chaque pas de temps.

public static int trackDate [5] [8760]

Tableau d'entier qui enregistre le numéro du pas de temps ainsi que la date (année, jour, mois, heure).

!!!! Attention !!!!

Les numéros qui définissent les limites des tableaux sont fixes et doivent être modifiées selon les paramètres de la simulation.

trackLocation [**nombre max d'individus**] [**nombre de pas de temps**]

trackDate [5] [**nombre de pas de temps**]

Méthodes

public void doProcess (PikesGroup group)

Procédure qui, à chaque pas de temps, enregistre la date courante, le nombre de pas dans le tableau trackDate, et les positions de chaque individu sont enregistrés dans le tableau trackLocation.