**Chronogram**

# A faire 2019.04.29

1. Procédure de conversion couche Qgis vers grid SimMasto
2. Utilitaire pour charger et empiler des couches dans une seule grid. Fait
3. Zoom complet autour du marché
4. Finaliser le comportement de la souris avec terrier, mangé, « énergie, odeur »
5. Tique (avec Jean)
6. Bactérie
7. Chat / Homme

# Commit :

C\_RodentDomestic2 : temporaire Changement de wall following with reactive behaviour en cours...

C\_ContextCreator : Ajout d'une fonction générique de générateur de boolean

C\_Landascape , A\_Animal : changement de canInteractWithThing par canperceiveThing

C\_ComputeRasterWithList : Ajout d'un compositeur de raster

C\_VariousUtilities : ajout des fonctions

- distancebetweenTwoPoint

- sum vectors

- substractvectors

- isCollinearVector

C\_SoilCellCity : ajout de la classe

# Bugs

Correction du bug sur le choix de la direction (left, right, forward, backward) en utilisant le filtre de mur.

Correction du bug sur le choix de la zone près du mur en utilisant le filtre de mur.

# Codes

/\*\* Use rodent perceived things to verify if it is in rebound zone and return the result \*/

**public** **boolean** isInReboundBand(TreeSet<I\_SituatedThing> perceivedThings) {

**for** (I\_SituatedThing oneThing : perceivedThings) {

**if** (((A\_Container) oneThing).getAffinity() == *WALL*) {

Coordinate thingCoordinate = oneThing.getCoordinate\_Umeter();

**if** (thingCoordinate.x > **this**.getCoordinate\_Umeter().x) {

**if** ((thingCoordinate.x - **this**.getCoordinate\_Umeter().x) < C\_Parameters.*CELL\_WIDTH\_Umeter*) **if** ((**this**.getCoordinate\_Umeter().x > (thingCoordinate.x

- *MAX\_VALUE\_OF\_REBOUNDZONE* - *MIN\_VALUE\_OF\_REBOUNDZONE*)) && (thingCoordinate.x > (**this**.getCoordinate\_Umeter().x + *MIN\_VALUE\_OF\_REBOUNDZONE*))) **return** **true**;}

**else** {

**if** (thingCoordinate.x < **this**.getCoordinate\_Umeter().x) **if** ((**this**.getCoordinate\_Umeter().x - thingCoordinate.x) >= (C\_Parameters.*CELL\_WIDTH\_Umeter* + *MIN\_VALUE\_OF\_REBOUNDZONE*)) {

**if** ((**this**.getCoordinate\_Umeter().y - thingCoordinate.y) >= (C\_Parameters.*CELL\_WIDTH\_Umeter* + *MIN\_VALUE\_OF\_REBOUNDZONE*)) {

**if** ((**this**.getCoordinate\_Umeter().x <= (thingCoordinate.x + C\_Parameters.*CELL\_WIDTH\_Umeter* + *MIN\_VALUE\_OF\_REBOUNDZONE* + *MAX\_VALUE\_OF\_REBOUNDZONE*))

&& (**this**.getCoordinate\_Umeter().y <= (thingCoordinate.y + C\_Parameters.*CELL\_WIDTH\_Umeter*

+ *MIN\_VALUE\_OF\_REBOUNDZONE* + *MAX\_VALUE\_OF\_REBOUNDZONE*))) **return** **true**;

}

**else** **if** (((thingCoordinate.y - *MIN\_VALUE\_OF\_REBOUNDZONE*) > **this**.getCoordinate\_Umeter().y)

&& (**this**.getCoordinate\_Umeter().y > (thingCoordinate.y - *MIN\_VALUE\_OF\_REBOUNDZONE* - *MAX\_VALUE\_OF\_REBOUNDZONE*))) **return** **true**;

}

**else** {

**if** (((thingCoordinate.y - *MIN\_VALUE\_OF\_REBOUNDZONE*) > **this**.getCoordinate\_Umeter().y)

&& (**this**.getCoordinate\_Umeter().y > (thingCoordinate.y - *MAX\_VALUE\_OF\_REBOUNDZONE* - *MIN\_VALUE\_OF\_REBOUNDZONE*))) **return** **true**;

**else** **if** (((**this**.getCoordinate\_Umeter().y - thingCoordinate.y) >= (C\_Parameters.*CELL\_WIDTH\_Umeter* + *MIN\_VALUE\_OF\_REBOUNDZONE*))

&& (**this**.getCoordinate\_Umeter().y <= (thingCoordinate.y + C\_Parameters.*CELL\_WIDTH\_Umeter* + *MIN\_VALUE\_OF\_REBOUNDZONE* + *MAX\_VALUE\_OF\_REBOUNDZONE*))) **return** **true**;

}

}

}

}

**return** **false**;

}

**if** (**this**.getDesire().equals(***REPRODUCTION***)) {

**if** (**this**.realTarget == **null**) {

I\_SituatedThing oneThing = **this**.chooseClosest(**this**.isSeeingInterestedThings(alternatives));

**if** (**this**.canInteractWithThing(oneThing)) {

**if** (**this**.getAnimalsTargetingMe().isEmpty() || !(**this**.getAnimalsTargetingMe().contains(oneThing))) {

selectedThing = **this**.isObstacleBefore(oneThing);

**if** (selectedThing == **null**) {

selectedThing = oneThing;

**this**.realTarget = **null**;

}

**else** {

**this**.realTarget = oneThing;

**this**.setDesire(***GO\_ALONG***);

}

}

}

**else** {

perceivedWalls = **this**.chooseWallPerceived(alternatives);

**if** (perceivedWalls.isEmpty()) perceivedWalls = alternatives;

}

}

}

**Else**

/\*\* verify if interesting thing is seeing \*/

**public** TreeSet<I\_SituatedThing> isSeeingInterestedThings(TreeSet<I\_SituatedThing> perceivedThings) {

TreeSet<I\_SituatedThing> seeingThings = **new** TreeSet<I\_SituatedThing>();

**for** (I\_SituatedThing oneThing : perceivedThings) {

**if** (oneThing **instanceof** A\_Animal) seeingThings.add(oneThing);

}

**return** seeingThings;

}

/\*\* verify if agent can go forward \*/

**public** **boolean** canGoForward() {

// double distance = this.distanceToTarget\_Umeter(this.getCoordinate\_Umeter(), this.myForward.getCoordinate\_Umeter());

// if (distance > (this.halfCellDiagonal()) && isWall(this.myForward)) {

// wallPosition = FORWARD\_TO\_AGENT;

// return true;

// }

// else

**return** (**this**.myFront != **null**) && !**this**.isWall(**this**.myFront);

}

List raster parameter :

String[] list\_URL = ((String) C\_Parameters.parameters.getValue("RASTERLIST\_FILE")).split(",");

// for (int i = 0; i < list\_URL.length; i++) {

// list\_URL[i] = RASTER\_PATH + "20180814\_RasterDodel2/" + list\_URL[i];

// }

// C\_Parameters.RASTERLIST\_URL = list\_URL;

Gestion raster list :

// **TODO** MS 08.2018 new constructor to load raster files

**public** C\_Landscape(Context<Object> context, String[] urlList, String gridValueName, String continuousSpaceName) {

**if** (urlList.length != 0) {

**int**[][] matrixRead;

**int**[][] matrixToCompare;

**if** (***RASTER\_MODE***.compareTo("ascii") == 0) matrixRead = C\_ReadRaster.*txtRasterLoader*(urlList[0]);

**else** matrixRead = C\_ReadRaster.*imgRasterLoader*(urlList[0]);

**for** (**int** i = 1; i < urlList.length; i++) {

**if** (***RASTER\_MODE***.compareTo("ascii") == 0) matrixToCompare = C\_ReadRaster.*txtRasterLoader*(urlList[i]);

**else** matrixToCompare = C\_ReadRaster.*imgRasterLoader*(urlList[i]);

**for** (**int** j = 0; j < matrixRead.length; j++)

**for** (**int** k = 0; k < matrixRead[0].length; k++) {

**if** (matrixRead[j][k] != matrixToCompare[j][k] && matrixToCompare[j][k] != 0) matrixRead[j][k] = matrixToCompare[j][k];

}

}

// IDENTIFY DIMENSIONS

**this**.dimension\_Ucell = **new** Dimension(matrixRead.length, matrixRead[0].length);

System.***out***.println("C\_Landscape: dimensions dim-ucell: width = " + dimension\_Ucell.width + " height = " + dimension\_Ucell.height);

// INIT EMPTY GRID OF CONTAINERS

**this**.grid = **new** I\_Container[(**int**) dimension\_Ucell.getWidth()][(**int**) dimension\_Ucell.getHeight()];

System.***out***.println("C\_Landscape: Initialized empty grid of containers");

// SET GRID VALUE LAYER WITH DIMENSIONS, PUT IT IN THE CONTEXT

**this**.gridValueLayer = **new** GridValueLayer(gridValueName, **true**, **new** repast.simphony.space.grid.WrapAroundBorders(), dimension\_Ucell.width,

dimension\_Ucell.height);

context.addValueLayer(**this**.gridValueLayer);

System.***out***.println("C\_Landscape: dimensions gridValueLayer: " + gridValueLayer.getDimensions());

// SET CONTINUOUS SPACE

ContinuousSpaceFactory continousfactory = ContinuousSpaceFactoryFinder.*createContinuousSpaceFactory*(**new** TreeMap<String, Object>());

**this**.continuousSpace = continousfactory.createContinuousSpace(continuousSpaceName, context, **new** SimpleCartesianAdder(),

**new** WrapAroundBorders(), gridValueLayer.getDimensions().getWidth(), gridValueLayer.getDimensions().getHeight());

System.***out***.println("C\_Landscape(): dimensions continuous space:" + continuousSpace.getDimensions());

System.***out***.println("C\_Landscape(): " + C\_Parameters.*CELL\_WIDTH\_Umeter* + " metres in one cell");

// Fill both (!) gridValueLayer and C\_SoilCell matrices with the value read in the raster

**this**.createGround(matrixRead);

}

}

## 20190813-Ancien colorMap Dodel2 Full:

**public** Map<Integer, Color> colorMapDodel2\_old(Map<Integer, Color> colorMap) {

colorMap = **new** HashMap<Integer, Color>();

// R G B red green blue

colorMap.put(0, **new** Color(243, 240, 233)); //  ground cell

colorMap.put(1, **new** Color(155, 155, 155)); // workshop

colorMap.put(2, **new** Color(255, 204, 153)); // Bakery

colorMap.put(3, **new** Color(255, 60, 255)); // Shop

colorMap.put(4, **new** Color(255, 245, 24)); // Office

colorMap.put(5, **new** Color(243, 240, 233)); // Hut 242, 226, 9

colorMap.put(6, **new** Color(237, 180, 23)); // Room

colorMap.put(7, **new** Color(0, 0, 0)); // Hairdresser 255, 0, 0

colorMap.put(8, **new** Color(197, 216, 21)); // Kitchen

colorMap.put(9, **new** Color(197, 216, 21)); // Dibiterien

colorMap.put(10, **new** Color(0, 255, 64)); // Enclosure

colorMap.put(11, **new** Color(242, 255, 99)); // Garage

colorMap.put(12, **new** Color(17, 178, 151)); // Garden

colorMap.put(13, **new** Color(103, 200, 255)); // Laboratory

colorMap.put(14, **new** Color(0, 128, 128)); // Magazin

colorMap.put(15, **new** Color(0, 251, 251)); // Mosque

colorMap.put(16, **new** Color(255, 0, 0)); // Mill

colorMap.put(17, **new** Color(247, 210, 240)); // Hardware store

colorMap.put(18, **new** Color(0, 200, 0)); // Restaurant

colorMap.put(19, **new** Color(128, 64, 64)); // Ruined

colorMap.put(20, **new** Color(0, 0, 255)); // Class

colorMap.put(21, **new** Color(255, 128, 64)); // Living room

colorMap.put(22, **new** Color(184, 96, 37)); // Tangana

colorMap.put(23, **new** Color(0, 128, 255)); // Bulding

colorMap.put(24, **new** Color(0, 0, 0)); // Wall

colorMap.put(25, **new** Color(221, 239, 247)); // House

colorMap.put(26, **new** Color(180, 177, 92)); // Market

colorMap.put(27, **new** Color(0, 0, 128)); // National Road

colorMap.put(28, **new** Color(128, 64, 64)); // Track

**return** colorMap;

}