Aggregation of Spatial Entities with CORMAS

1. Definitions

The generic term of "aggregate" stands for a compound spatial entity which constitutive elements are verifying a constraint of contiguity. With CORMAS, there are basically three elementary operations allowing to create aggregates.

- **Conditional selection**. If all the cells of the spatial grid verify a given condition of aggregation, then the whole spatial grid will represent a single and unique aggregate, which may not be very interesting. This operation is used when a certain number of elementary entities will not verify the condition, and therefore will not belong to any aggregate. Consequently, the aggregates produced by this operation cannot have contact points, otherwise they would merge (cf. figure 1a).
- **Partition**. There are two possibilities : either the elements are aggregated according to the values of one or more attributes, either the elements are aggregated according to a number of horizontal subdivisions of the spatial grid and a number of vertical subdivisions of the spatial grid. The first possibility allows to define "excluding" values of the attributes, which may lead to an incomplete partition of the whole grid, but usually, the aggregates produced are joined (cf. figure 1b).
- **Expansion from « seeds »**. This recursive operation allows to aggregate elements which are not yet belonging to any other aggregate of the same kind. A pre-defined number of components may be assigned to each aggregate, as shown in figure 1c (25 components per aggregate).

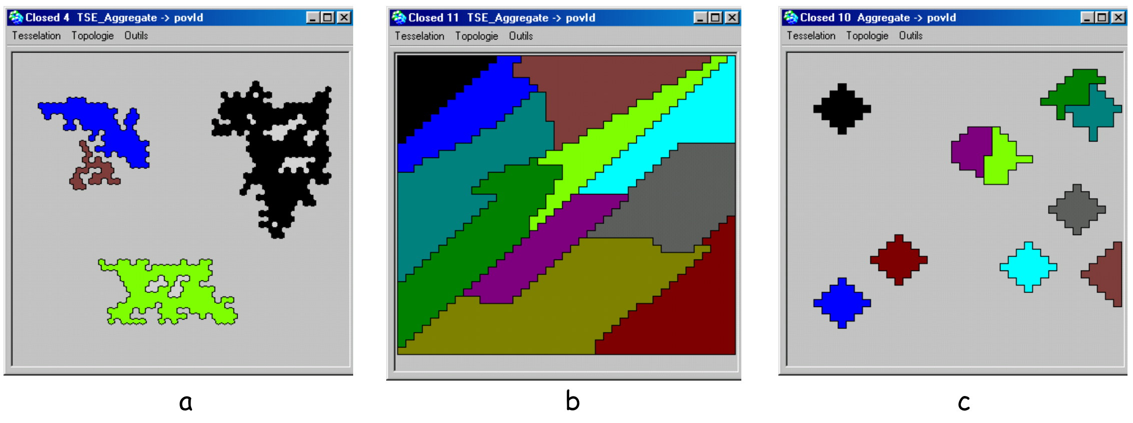
Closed 4 TSE_Aggregate -> povId
Tesselation Topologie Outils
Closed 11 TSE_Aggregate -> povId
Tesselation Topologie Outils
Closed 10 Aggregate -> povId
Tesselation Topologie Outils
a
b
c

Figure 1 : Examples of aggregates obtained by :
conditional selection (a), partition (b), expansion (c)

2. CORMAS Methods

All the methods for the creation of regular polygons (the spatial grid is a rectangular matrix made of squared or hexagonal cells) are located in a protocol « *regular polygons – instance creation* » of the `SpaceModel` class. These methods are self-sufficient : all the instructions that were accompanying the previous versions of the aggregation methods (among them, the ones to allow the display on the grid, `setOuline` and `setImage`) are not needed anymore. By now, the execution of these methods automatically realizes the affectation of the newly created aggregates within the attribute of the model named "theCompoundEntitys", by overwriting the previous ones : it is not an update but a complete building process.

2.1. Conditional selection

2.1.1. Basic method

The basic method to creates aggregates by conditional selection is used in any CORMAS model with the following instruction :

```
self spaceModel
      setAggregates: CompoundEntity
      from: BaseEntity
      verifying: aBlock.
```

BaseEntity and CompoundEntity are class names. CompoundEntity has to inherit from SpatialEntityAggregate. BaseEntity most often inherits from SpatialEntityElement, but it is possible to create aggregates made of aggregates. The last argument of the method is a smalltalk block used to specify the condition to aggregate the components. It takes the general following form : `[:c | c <condition>]`, where c represents any component.

2.1.2. Variants

There are variants allowing to give a minimum number of components and/or the "type" (an attribute of the SpatialEntitySet class) to be assigned to the newly created aggregates.

```
self spaceModel
    setAggregates: CompoundEntity
    from: BaseEntity
    verifying: aBlock
    minimumSize: n
```

```
self spaceModel
    setAggregates: CompoundEntity
    from: BaseEntity
    verifying: aBlock
    type: aValue
```

n is an integer number, aValue is a value of any kind (string, symbol, number, etc…) to be assigned to the "type" attribute of the newly created aggregates.

An example of conditional aggregation with minimum size is given in the **TSE** (**T**est **S**patial **E**ntities) model. Load the environment « *3forests.env* », then select initForests as the initialization method, and stepForests: as the control method. This scenario proposes a dynamics defined at two spatial levels. One may look at what is happening either at the cellular level (select the point of view "povTree" from the contextual menu of the spatial grid) or at the aggregated level (select either the "povSize" point of view, which is relating the darkness of green to the size of the aggregates, either the "povId" point of view, which is assigning a different color for each aggregate, as in figure 1a).

2.1.3. Importing/Exporting

It is possible to export, in an ascii format, the aggregates and their neighborhood. This option may be useful when the aggregation process is long to process: the operation is performed only once, then the result is exported. To have an idea of how writing and reading the corresponding files, look at the **TSE** model (methods "exportNeighbors", "importNeighbors", "exportForests" and "importForests").

2.2. Partition

2.2.1. Basic method

The basic method to creates aggregates by partition is used in any CORMAS model with the following instruction :

```
self spaceModel
      setAggregates: CompoundEntity
      from: BaseEntity
      attribute: attributeName.
```

The instances of the `CompoundEntity` class are created by aggregating instances of the `BaseEntity` class that are contiguous and that are holding the same value for the specified attribute. The attribute name, given as the last argument of the method, has to be a "symbol" (example : `#landUse`). Instances of the `BaseEntity` class holding undefined (nil) value for the attribute will not be taken into account by the operation.

An example of partition based on the different values hold by an attribute is given in the **TSE** model (see the result in figure 1b). Load the environment "*partitions.env*". Select one of the two aggregated points of view ("`povSize`" or "`povId`"). Select "`initPartitionsFromMap`" as the initialization method, and "`step:`" as the control method (note that in this example, once the creation of the aggregates is achieved by executing the initialization method, the newly created aggregates remain the same over time, so the "`step:`" method is inefficient here).

2.2.2. Variants

There is one variant allowing to specify a special defined value of the attribute to exclude components from the aggregation process (note that a "nil" value is always preventing the aggregation, as mentioned earlier) :

```
self spaceModel
      setAggregates: CompoundEntity
      from: BaseEntity
      attribute: attributeName1
      excludingValue: aValue.
```

To proceed the partition according to a pair of attributes, there is a specific method. To be aggregated, contiguous instances of the "`BaseEntity`" class should have the same values for both "`attributeName1`" and "`attributeName2`" :

```
self spaceModel
      setAggregates: CompoundEntity
      from: BaseEntity
      attribute: attributeName1
      attribute: attributeName2.
```

To perform a partition being a kind of regular subdivision of the spatial grid, there is a specific method that looks like :

```
self spaceModel
      setAggregates: CompoundEntity
      from: BaseEntity
      horizontalDividers: i
      verticalDividers: j.
```

`i` and `j` are integer numbers allowing to divide along horizontal and vertical directions the rectangular grid into smaller rectangles that will be the newly created aggregates. In the **TSE** model, there is an example of partition of a spatial grid made of 41x41 cells by subdividing it in 5 along the horizontal direction and in 4 along the vertical direction (see the initialization method named "`setAggregatesFifthFourth`"). The execution of this method leads to the creation of 20 aggregates (cf. figure 2). Note that this method automatically adjusts the width of the aggregates to the integer value of the division of the total number of columns of the spatial grid by `i` (8 in our example); in a similar way, the height of the aggregates is set to the integer value of the division of the total number of rows of the spatial grid by `j` (10 in our example). The remaining cells (1 for the both directions in our example) become additional components of the last column and/or last row of newly created aggregates.

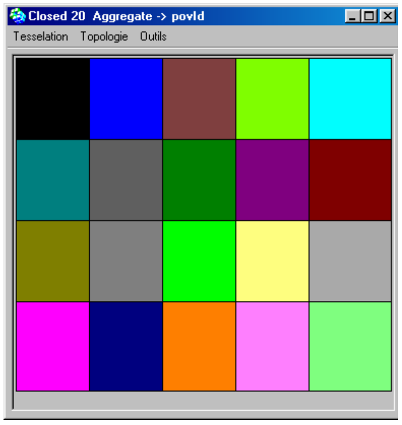

Figure 2 : example of partition by regular subdivision

Among the 20 aggregates shown in figure 2, 12 are 8x10 rectangles, 4 are 8x11 rectangles, 3 are 9x10 rectangles and finally the one located at the lower-right part of the grid is a 9x11 rectangle.

2.3. Expansion from seeds

CORMAS is providing a set of methods to create aggregates by expansion from seed-components.

2.3.1. Basic method

All the methods corresponding to this kind of operation are first creating single-component aggregates (the kernel component, also referred to as the "seed") :

```
self spaceModel
     setSingletonAggregates: CompoundEntity
     fromSeeds: aCollec.
```

The execution of the above method will effectively create `n` aggregates, `n` being the number of seeds, which is equal to the size of `aCollec`, the collection given as the last argument. An aggregate is expanding by incorporating instances of the same class than the seeds that are belonging to its surroundings and that have not yet been incorporated to another aggregate being an instance of `CompoundEntity`. The corresponding instruction is written as follow :

```
self spaceModel swell: CompoundEntity.
```

For a progressive expansion process, one surroundings after another, at each time-step of simulation, the above statement may be used directly within the step method of the model (in the **TSE** model, select `setSingletonAggregatesFromRandomSeeds` as the initialization method and `swellAggregates` as the control method).

For a complete expansion in a single round, as the aggregates are created, the following method is available :

```
self spaceModel
      setAggregates: CompoundEntity
      fromSeeds: aCollec.
```

As a result of this statement, the partition of the spatial grid is completed (figure 3 shows an example produced with the **TSE** model, by selecting `setAggregatesFromGivenSeeds` as initialization method).

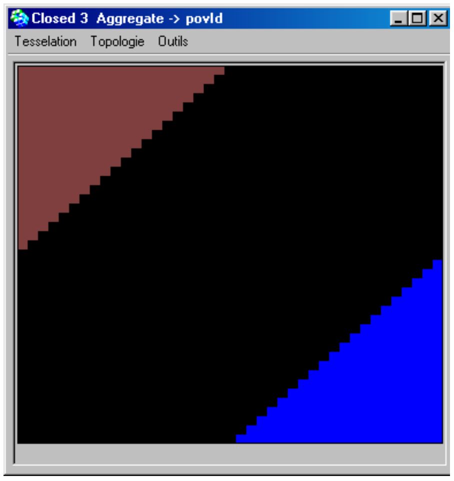

Figure 3 : example of a complete partition from the expansion of 3 seeds (top left, bottom-right and central locations)

2.3.2. Variants

To constraint the process of expansion, one may specify an additional condition to the selection of new components in the surroundings of the newly created aggregates. The corresponding method looks like :

```
self spaceModel
      swell: CompoundEntity
      verifying: aBlock.
```

The conditional block has to be written as usual like this : `[:c | c <condition>]`. In the same way, to expand in a single round to the whole spatial grid the aggregates, with an additional condition set on the selection of the components, there is a specific method :

```
self spaceModel
      setAggregates: CompoundEntity
      fromSeeds: aCollec
      verifying: aBlock.
```

It is possible to stop the expansion process when the aggregates are reaching a given size (a number of components). The definition of the method stands as follow:

```
self spaceModel
      setAggregates: CompoundEntity
      fromSeeds: aCollec
      sizeDistribution: aDistribution.
```

The last argument of this method is a vector of pairs of integer numbers. The first element of each pair gives the number of aggregates, the last one indicates the number of components. By using the Smalltalk way to create arrays in literal form, a vector of pairs of integer numbers looks like : `#(#(`N_1 X_1`) #(`N_2 X_2`) … #(`N_p X_p`))`. Let's imagine you would like to create 9 aggregates, 5 made of 40 components, 3 made of 50 components and 1 made of 100 components. Then you should use `#(#(5 40) #(3 50)(1 100))` as the last argument of the method presented above. You have to make sure that the size of the collection of seeds is matching the sum of all N_i.

Figure 1c gives an example of creation of 10 aggregates with a given size equal to 25 components from 10 seeds randomly set within a 41*41 spatial grid (see `setAggregatesFromRandomSeeds in the` **TSE** model).