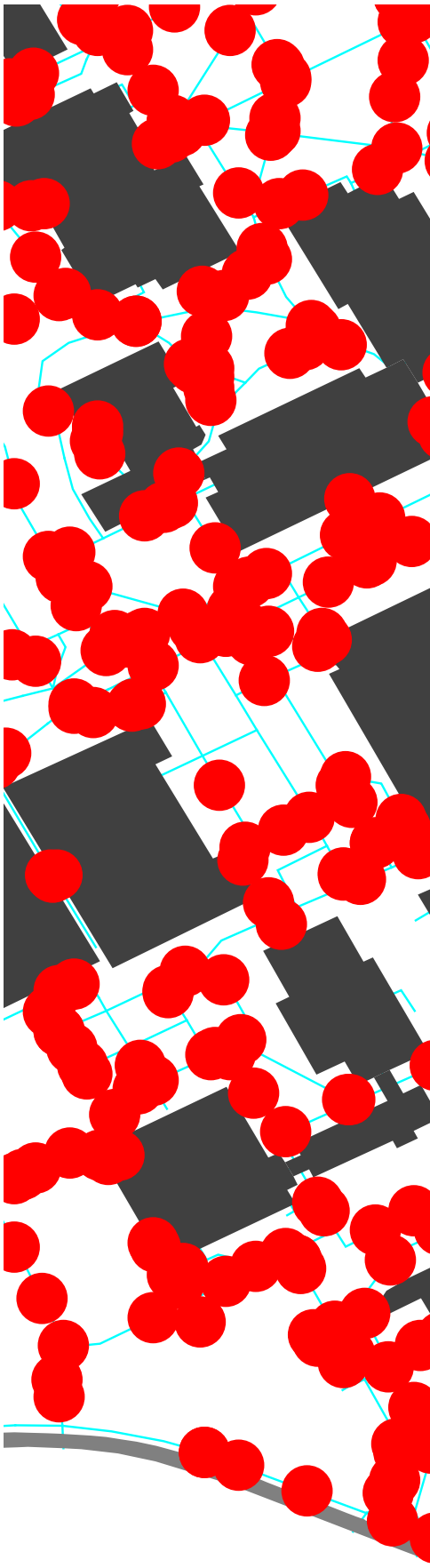
The GeoMason Cookbook

Mark Coletti
Department of Computer Science
George Mason University

Zeroth Edition
Online Version 0.1
January, 2013



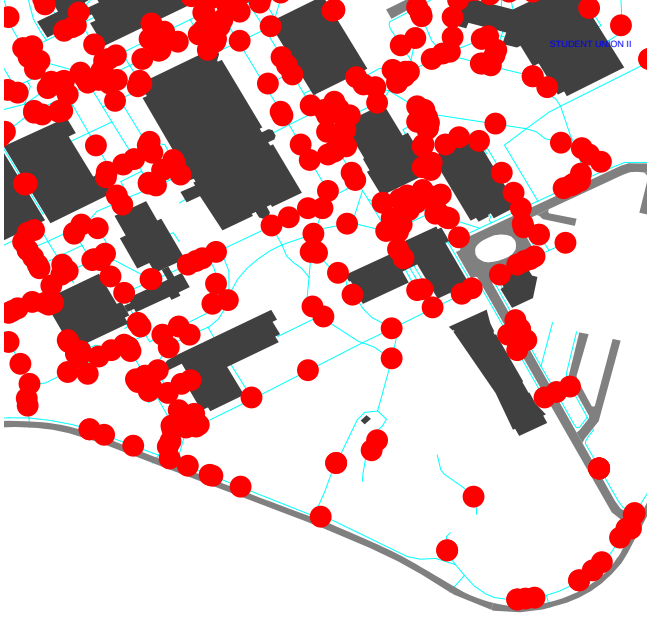
Where to Obtain GeoMason
http://cs.gmu.edu/~eclab/projects/mason/extensions/geomason/

Thanks to Sean Luke, Andrew Crooks, Keith Sullivan

Get the latest version of this document or suggest improvements here:

http://cs.gmu.edu/∼eclab/projects/mason/extensions/geomason

Contents



Chapter 1

Introduction

1.1 What is GeoMason?

MASON is a sophisticated multi-agent discrete event simulation library. Unfortunately it does not natively support geospatial data, which means that such support has to be hand crafted by users. Providing such support can be error prone and tedious to implement.

Fortunatey, there exists a MASON extension, GeoMason, that imbues MASON with some limited geospatial awareness. With GeoMason one is able to load, display, and manipulate data that is, in some way, grounded to the Earth's surface.

1.2 The Motivation Behind this Book

This "cookbook" provides a set of "recipes" for using GeoMason and is meant to fill in the documentation gap between javadoc generated API documentation and the GeoMason technical report. I felt that a full blown manual would do little more than elaborate on what was found in the API documentation. That is, a manual would explain the "what" and still leave the "how" largely unanswered. The "cookbook" format is popular in technical literature because it provides an accessible "use case" oriented style. In the cookbook format, it's easy for a reader to find a use case, or "recipe," that best matches what they want to do, and then use that recipe as inspiration for their own tailored solution. And so I felt that this format best addressed the "how".

For those unfamiliar with the "Cookbook" format, a technical "Cookbook" is comprised of *recipes*. Again, these recipes embody a simple use case, and is summarized by its title such as "Moving Agents Along a Path." Each recipe is comprised of a simple problem statement, a simple solution statement optionally followed by code examples, and then lastly followed by a discussion section that elaborates on the problem and posed solution.

1.3 What this Book Is Not

Unfortunately I have had to make a few assumptions with this book, which I hope are reasonable. In particular, I presume that readers are familiar with MASON, Geospatial Information Systems (GIS), and, naturally, java. This Cookbook is not intended to be a MASON, GIS, or java primer. If you are unfamiliar with MASON, I recommend reading its manual [1] and working through its tutorials. If you need to bone

[1] `http://cs.gmu.edu/~eclab/projects/mason/manual.pdf`

up on GIS, there exist many resources to get you up to speed. The Wikipedia page for GIS is a good online starting point.[2] The book *Geographic Information Systems and Science* by Goodchild et al is also an accessible read.[3] Naturally there exist numerous online resources on java.

1.4 Book Organization

I've divided the book into five chapters of which the first one you're reading now. The second chapter provides I/O related recipes; i.e., recipes for reading and writing geospatial data using GeoMason. The third chapter is by far the most complex — and, even then I confess it probably falls short of some needs. This chapter deals with *using* GeoMason to some end, whether it's as basic as finding nearby objects or moving agents along paths or having agents follow gradients. I hope to in the future expand some of the leaner recipes, particularly those involving shortest path computations, and to add new ones, especially ones suggested by the GeoMason community. The next chapter involves recipes for displaying geospatial data such as rendering boundary lines over grid data. And, finally the last chapter has a couple recipes for common GeoMason problems; hopefully this chapter won't expand much over time!

[2] `http://en.wikipedia.org/wiki/Geographic_information_system`

[3] Longley, P.A., Goodchild, M.F., Maguire, D.J. and Rhind, D.W. (2005) Geographic Information Systems and Science. Chichester: Wiley. 2nd edition.

Chapter 2

Reading and Writing Geospatial Data

This chapter covers recipes for reading and writing vector and grid based geospatial data.

2.1 Reading Geospatial Data

This section covers reading geospatial data into MASON using GeoMason.

2.1.1 Reading a Shape File

Problem

You want to read vector geospatial data stored in a Shape file.

Solution

Create a GeomVectorField and use ShapeFileImporter.read() to load data into it.

```
// WIDTH and HEIGHT correspond to arbitrary display dimensions
private final int WIDTH = 300;
private final int HEIGHT = 300;
GeomVectorField vectorField = new GeomVectorField(WIDTH, HEIGHT);

try {
   ShapeFileImporter.read("file:foo.shp", vectorField);
} catch (FileNotFoundException ex)
{  /* handle exception */  }
```

Discussion

Though there exist other GeoMason classes capable of reading Shape files — GeoToolsImporter and OGRImporter — the native GeoMason shape file importer, ShapeFileImporter, is recommended, especially given that it has no third party dependencies as the other importer classes do.

Given the general static nature of shape files, the above code snippet is likely to be in a SimState subclass constructor. Alternatively you may place it in the start() though be mindful that means that the shape file will be loaded again each time the simulation is restarted.

Note that the units of the loaded vector layer will be those of the underyling coordinate reference system. So if the shape file is in meters, such as is found in data in Universal Transverse Mercator

(UTM), then all loaded geometry will similarly be in meters. Also note that GeoMason uses the JTS Topology Suite to store all the geometry. JTS Topology Suite uses a flat Cartesian plane for all points; so be aware of this when loading data from a non-planar reference system. That is, if you naïvely load, say, native lat/lon data, which corresponds to coordinates along a ellipsoid, that you will have introduced distortions in the implicit projection you have just done to a 2D plane. Moreover, these distortions will be more pronounced for large surface areas.

Note that WIDTH and HEIGHT correspond to the display dimensions and are used to help properly scale the rendered field when panning and zooming. Note that at some point the need for specifying the field width and height will be unnecessary.

2.1.2 Reading Multiple Vector Layers

Problem

You want to read in more than one thematic layer of vector data.

Solution

After ensuring that each layer uses the same coordinate reference system, read in each layer, and then synchronize the minimum bounding rectangles (MBR) for all the layers.

```
GeomVectorField firstVectorField = new GeomVectorField(WIDTH,HEIGHT);
GeomVectorField secondVectorField = new GeomVectorField(WIDTH,HEIGHT);
GeomVectorField thirdVectorField = new GeomVectorField(WIDTH,HEIGHT);

try {
   ShapeFileImporter.read("file:foo.shp", firstVectorField);
   ShapeFileImporter.read("file:bar.shp", secondVectorField);
   ShapeFileImporter.read("file:baz.shp", thirdVectorField);
} catch (FileNotFoundException ex)
{  /* handle exception */  }

Envelope globalMBR = firstVectorField.getMBR();

globalMBR.expandToInclude(secondVectorField.getMBR());
globalMBR.expandToInclude(thirdVectorField.getMBR());

firstVectorField.setMBR(globalMBR);
secondVectorField.setMBR(globalMBR);
thirdVectorField.setMBR(globalMBR);
```

Discussion

It is possible that the disparate shape files may have different coordinate reference systems, as can happen if the shape files came from different sources. It is vitally important to ensure that all the layers have the same coordinate reference system before being loaded into GeoMason. For example, a vector layer that uses lat/lon coordinates will have radically different geometry values from another vector layer that uses UTM even though they may cover the same area on Earth. Essentially, GeoMason is not a GIS so it will not do on-the-fly projections of the data. Users can use a real GIS tool, such as QuantumGIS [1], to manually reproject data prior to loading into GeoMason.

[1] http://www.qgis.org/

It is important to ensure that all the layers have the same MBR otherwise they will not align properly when displayed. Naturally, this is optional if you do not intend on rendering the layers. Regardless it would be prudent to do so anyway on the chance that later you change your mind and want to see the GeomVectorFields. The highlighted lines 12-19 show how to synchronize the MBRs between loaded GeomVectorFields. Basically, you get the MBR of the first GeomVectorField, expand it to include the area of the MBRs for the remaining GeomVectorFields, and then set them all to the one all-inclusive MBR. Note that the successive calles to expandToInclude() monotonically increase the returned MBRs.

As noted in recipe 2.1.1, given the general static nature of shape files, this code snippet is likely to be done in the SimState constructor; however, again, the layers could also be loaded via start(), though that means loading the shape files every time the simulation is re-run.

2.1.3 Reading a Shape File and Some of Its Attributes

Problem

You want to read a Shape file and only some of its associated attributes.

Solution

Read in a shape file as in recipe 2.1.1, but specify the desired attributes by creating a Bag of Strings containing attribute names, and then passing that Bag to ShapeFileImporter.read().

Reading Attributes

```
GeomVectorField vectorField = new GeomVectorField(WIDTH,HEIGHT);

Bag desiredAttributes = new Bag();
desiredAttributes.add("NAME");
desiredAttributes.add("TYPE");

try {
   ShapeFileImporter.read("file:foo.shp", vectorField, desiredAttributes);
} catch (FileNotFoundException ex)
{  /* handle exception */  }
```

Each spatial object is wrapped in a MasonGeometry object which, in turn, also stores any associated attributes. Use the appropriate MasonGeometry get*Attribute() method to retrieve the attribute value.

Using Attributes

```
Bag geometries = vectorField.getGeometries();

for (int i = 0; i < geometries.size(); i++)
{
    MasonGeometry geometry = (MasonGeometry) geometries.objs[i];

    int type = geometry.getIntegerAttribute("TYPE");
    String name = geometry.getStringAttribute("NAME");
}
```

Discussion

Most shape files have an associated set of attributes describing each feature. For example, buildings

will have names, roads will have a number of lanes, bridges will have a type, and so on. These attributes can be strings, numbers, or boolean values.

By default GeoMason will not load any associated attributes — if you want any attributes you will have to ask for them by name. You do this by filling a Bag with strings of desired attribute names, and then passing that Bag to a ShapeFileImporter.read() invocation, as shown in the highlighted lines 3-5 and 8 in the code example "Reading Attributes." This will then load each MasonGeometry object that corresponds to each spatial entity with a set of attribute/value pairs. These attributes can be later retrieved with an appropriate call to getStringAttribute(), getIntegerAttribute(), or getDoubleAttribute(); you can also invoke getAttribute() to retrieve the value object directly. An example of using these methods is shown in the code snippet "Using Attributes."

Unfortunately this does mean you have to know ahead of time the available attributes and their respective names. If you do not know the available attributes, you can use a GIS such as QuantumGIS or ArcGIS to discover the attribute names.

2.1.4 Reading an ARC/Info ASCII Grid File

Problem

You want to read an Arc/Info ASCII Grid file.

Solution

Create a GeomGridField and an InputStream opened on the grid file, then use ArcInfoASCGridImporter() to load the data into the grid field from the open input stream.

```
GeomGridField gridField = new GeomGridField();

InputStream inputStream = new FileInputStream("foo.asc");
ArcInfoASCGridImporter.read(inputStream, GridDataType.INTEGER, gridField);
```

Discussion

Essentially a GeomGridField is a wrapper round a MASON Grid2D object that imbues some limited geospatial characteristics – essentially it's a georeferenced MASON Grid2D.

ArcInfoASCGridImporter.read() will use one of two Grid2D MASON subclasses, IntGrid2D or DoubleGrid2D, depending on whether integer or real-value data is used, respectively. Unfornately GeoMason is not smart enough to figure out ahead of time which underlying MASON Grid2D subclass to use so you have to specify that via the GridDataType argument to ArcInfoASCGridImporter.read(); i.e., GridDataType.INTEGER for integer based grid data or GridDataType.DOUBLE for real-value based grid data.

Many times you can readily intuit the underlying data type of an ASC/Grid file. E.g., a grid of population values such as Landscan values [2] will likely use integers, wheres one of elevation postings, such as found in Digital Elevation Models, will likely use floating point values. However, if you do not know whether integers or floats are used in a given grid file, you can peep at it with a text editor to find out. If you have a UNIX-like command line, you can use "head -7" to look at the first seven lines of text in an ASC/Grid file. Regardless of how you look at the data, it should be readily apparently whether the file contains all integers or floats.

[2] http://www.ornl.gov/sci/landscan/

2.1.5 Reading a Mix of Grid and Vector Data

Problem

You want to read in multiple layers that are a mix of vector and grid geospatial data.

Solution

After ensuring that all the layers have the same coordinate reference system, read all the layers into GeomVectorField or GeomGridFields, as appropriate, and then synchronize their respective minimum bounding rectangles.

```
GeomVectorField vectorField = new GeomVectorField(WIDTH,HEIGHT);
GeomGridField gridField = new GeomGridField();

try {
   ShapeFileImporter.read("file:vector.shp", firstVectorField);

   InputStream inputStream = new FileInputStream("grid.asc");
   ArcInfoASCGridImporter.read(inputStream, GridDataType.INTEGER, gridField);
} catch (FileNotFoundException ex)
{  /* handle exception */  }

Envelope globalMBR = vectorField.getMBR();

globalMBR.expandToInclude(gridField.getMBR());

vectorField.setMBR(globalMBR);
gridField.setMBR(globalMBR);
```

Discussion

The same issues apply here as noted in recipe 2.1.2 — i.e., not only do the coordinate reference systems need to be identical between layers, but the MBRs also need to be synchronized. Also, as in recipe 2.1.4, you will have to specify the appropriate GridDataType that corresponds to type of data found in the grid file.

One typical scenario this recipe covers is overlaying political boundaries over grid data.

2.2 Writing Geospatial Data

This section covers recipes involving saving geospatial data from GeoMason constructs to files.

2.2.1 Writing a Shape File

Problem

You want to save a GeoMason vector field to a Shape file.

Solution

Use ShapeFileExporter.write() to save the vector field to a Shape file.

```
ShapeFileExporter.write("foo", vectorField);
```

Discussion

Obviously it doesn't make sense to write a shape file for shape files you've already read in because the data presumably hasn't changed. This recipe is, instead, useful for scenarios where you have a vector layer of data that you've created wholly within your simulation and wish to save so that you can load it into a proper GIS.

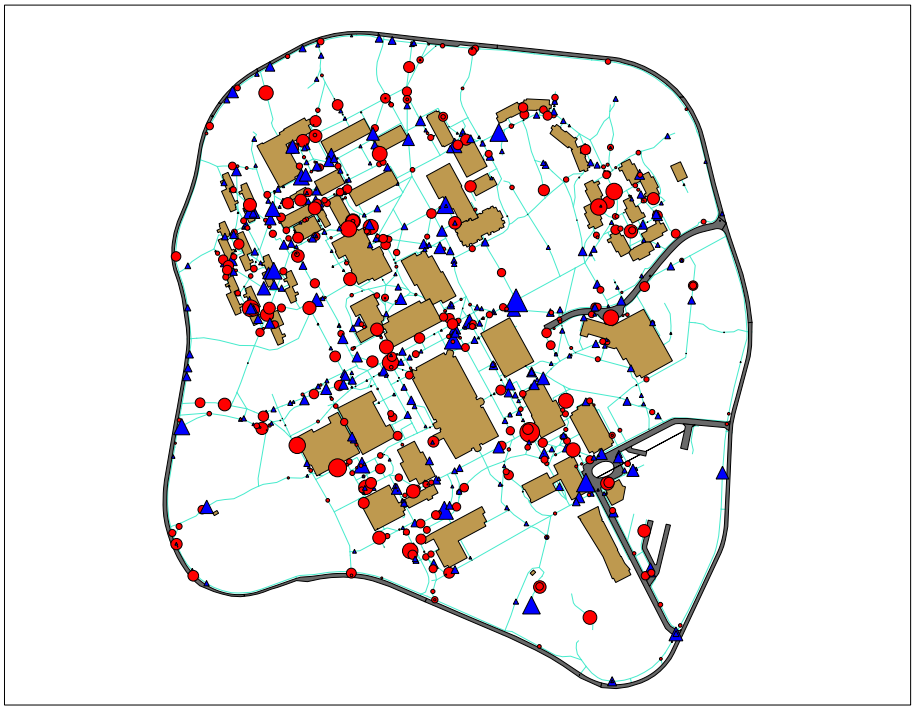


Figure 2.1 Snapshot of "Campus World" demo .

Consider the GeoMason "Campus World" demo that has agents moving along walkways.[3] The buildings, walkways, and roads were loaded from shape files, but the agents were created stochastically from within the simulation and stored in their own GeomVectorField. When the simulation ends a shape file describing these agents is written out to a shape file that can be loaded along with the original shape files for analysis. These agents have the following three attributes: their age, movement rate, and whether they are student or faculty. Fig. 2.1 depicts these shape files after they were loaded into a GIS with the faculty agents rendered as blue triangles, students as red dots, and their relative sizes scaled to their respective movement rates.

Shape files are not single files but are instead comprised of a few mandatory files with the extensions .shp, .dbf, and .idx. The first argument to ShapeFileExporter.write() specifies the file name prefix used to generate these files from the given GeomVectorField.

There is an optional shape file that contains the coordinate reference system and uses the file name extension .prj. This function does not write this file.[4] However, if the GeomVectorField was itself sourced from a shape file, then its corresponding .prj file can be copied over using the new file extension used in the call to write(). Alternatively, if the GeomVectorField was *not* read from a shape file, and so does not have a corresponding .prj file, you may still be in luck if you loaded other vector layers from shape files. If that's the case, you can likely arbitrarily use one of their .prj files.

[3]This demo is included with GeoMason, and can be found as sim.app.geo.campusworld

[4]However, a future incarnation of GeoMason may do so.

If you want to automatically save a snapshot of a GeomVectorField after each simulation run, you can place this call to write() within finish() inside your SimState subclass.

2.2.2 Writing an ARC/Info ASCII Grid File

Problem

You want to write GeoMason grid data to an ARC/Info ASCII Grid file.

Solution

Create a Writer for the grid file and use ArcInfoASCGridExporter.write() to write the GeomGridField.

```
try {
   BufferedWriter writer = new BufferedWriter( new FileWriter("foo.asc") );
   ArcInfoASCGridExporter.write(gridField, writer);
   writer.close();
} catch (IOException ex)  {
   /* handle exception */
}
```

Discussion

This recipe is useful for saving grid data from a simulation run such that you can later import it into a GIS for analysis.

Unlike calls to ArcInfoASCGridImporter.read(), you do not have to specify a data type. Instead, the call to ArcInfoASCGridExporter.write() automatically handles that detail for you.

As in recipe 2.2.1 you can place this snippet into finish() to automatically write a grid file when the simulation ends.

Chapter 3

Using Geospatial Data

In this chapter we discuss interacting with geospatial data using GeoMason. These kinds of interactions are mostly queries such as asking in what political boundaries an agent is located or determining nearby entities.

3.1 Determining What Political Entity an Agent is In

Problem

You have a simulation with polygons for boundaries that delineate political entities such as countries, counties, or voting districts. In that simulation you also have agents that move across these kinds of boundaries, and you would like to know the political entity in which that agent is located.

Solution

Assume that a GeomVectorField contains the political boundary polygons. If the agents are points in a GeomVectorField, then directly use their representative geometry to find what geometry covers them in the boundary vector field. Alternatively, if the agents are in a GeomGridField, then you can use the GeoMason grid field's toPoint() to convert from the grid coordinate to the underlying coordinate reference system, and then use that coordinate to find the boundary polygon that covers that point.

Agents as Vector Field

```
GeomVectorField boundaries = new GeomVectorField(WIDTH,HEIGHT);
GeomVectorField agents = new GeomVectorField(WIDTH,HEIGHT);

// ...

// Arbitrarily select first agent
MasonGeometry agent = (MasonGeometry) agents.getGeometries().objs[0];

// Bag should contain a single MasonGeometry object for polity in
// which agent is located.
Bag polity = boundaries.getCoveringObjects(agent.geometry);

if ( polity.isEmpty() ) {
   // the agent is not in any polity defined in boundaries
}
```

Agents as Grid Field

```
GeomVectorField boundaries = new GeomVectorField(WIDTH,HEIGHT);
GeomGridField agents = new GeomGridField();

// ...

// (x,y) denotes an arbitrary valid location within the grid field, agents
Point p = agents.toPoint(x, y);

// Again, Bag should contain a single MasonGeometry object for polity
// in which everything in that grid cell is located.
Bag polity = boundaries.getCoveringObjects(p);

if ( polity.isEmpty() ) {
   // the agent is not in any polity defined in boundaries
}
```

Discussion

Naturally all the various vector and grid layers should use the same underlying coordinate reference system and should have their respective minimum bounding rectangles (MBR) synchronized as per recipe 2.1.2.

Note that the isEmpty() check may return true even if you think that an agent should be in a given polity due to errors in the underylying data. E.g., the polygons between adjacent political entities may not align perfectly leaving little "void" gaps in which an agent can fall.

Also, the center points of grids are used as reference points to determine in which region a particular grid-based agent falls; some grid cells that overlap polygon edges may have center points that fall outside region polygons. This sometimes means that the wrong political region may be reported for a given grid cell. If more precision is required to avoid those scenarios, then instead of the toPoint() call, which is highlighted above on line 7, you can use toPolygon() to return the polygon outlining the corresponding grid cell. You can then use that polygon in the call to getCoveringObjects(). However, be aware that the returned Bag may contain multiple polygons corresponding to regions that cover that grid cell's rectangular coverage. In which case you must then figure out which of the regions the cell actually best covers.

3.2 Locating Nearby Geospatial Objects

Problem

You want to find all the objects within a certain distance from a specific thing.

Solution

You can use getObjectsWithinDistance() to find all objects within a certain distance from a given object.

```
GeomVectorField objects = new GeomVectorField(WIDTH,HEIGHT);
Point location; // assume later set to desired location

// ...

Bag nearestObjects = objects.getObjectsWithinDistance(location,distance);

if (nearestObjects.isEmpty()) { System.out.println("Nothing nearby"); }
```

Discussion

This recipe addresses finding geospatial objects in vector space. Naturally if you want to locate nearby objects in a grid, you can use the legacy MASON grid functions to locate nearby cells. [1]

3.3 Finding Adjacent Geospatial Objects

Problem

You want to find adjacent geospatial objects. For example, for a given country, you want to get a list of neighboring countries that share a common border.

Solution

This is very similar to recipe 3.2. Instead of using getObjectsWithinDistance() to find nearby objects, use getTouchingObjects(), instead, to find bordering objects.

```
GeomVectorField objects = new GeomVectorField(WIDTH,HEIGHT);
Point location; // assume later set to desired location

// ...

Bag nearestObjects = objects.getTouchingObjects(location);

if (nearestObjects.isEmpty()) { System.out.println("Nothing nearby"); }
```

Discussion

As for recipe 3.2., this is for the vector domain. Determining nearby objects in a grid is trivial to compute as they are just the adjacent grid cells.

Again, as the description stated, this recipe can be used to compute adjacent polities. Since those are generally static, you can compute the political entity adjacencies once when the simulation starts, and then store the values in a look-up table.

3.4 Moving Agents Along Paths

Problem

You have a path, such as a road or trail, along which you want to move an agent.

Solution

Presuming you are representing the paths as LineStrings, you can create a corresponding JTS Topology Suite LengthIndexedLine for each LineString.[2] LengthIndexedLine allows you to "index" along its length; i.e., you can get a JTS Coordinate for any position along a LengthIndexedLine. So agents that are represented as points (i.e., coordinates), can get their current point position by indexing along a given LengthIndexedLine. Each agent knows its index value for the current line it is on. To move, the agent increments or decrements that index by some value corresponding to a movement rate — the higher the value the faster the agent moves. It then gets the point for the position from its associated LengthIndexedLine and effects its movement by setting its point position to that value.

[1] Do note that the MASON nearest neighbor functions are currently in flux, so use with caution.

[2] If you have read in these line strings via a GeoMason importer, note that these LineStrings will be wrapped in MasonGeometry objects stored in a GeomVectorField.

The following code is for a simple agent that moves back and forth along a single line segment.

```
public class LineFollower implements Steppable
{
    // current position in 2D space
    private Coordinate position;

    // current index position along line
    private double currentIndex;

    // line along which agent is moving
    private LengthIndexedLine line;

    // how fast the agent moves along line
    private double moveRate = 1.0;

    public LineFollower(MasonGeometry geometry)
    {
        LineString lineString = (LineString) geometry.geometry;

        line = new LengthIndexedLine(lineString);

        // arbitrarily start at one end of the line
        currentIndex = line.getStartIndex();

        // update the 2D coordinate for that end of the line
        position = line.extractPoint(currentIndex);
    }

    @Override
    public void step(SimState state)
    {
        // If the next step takes us off the line, reverse course
        if (! line.isValidIndex(currentIndex + moveRate)) {
            moveRate *= -1;
        }

        currentIndex += moveRate;

        position = line.extractPoint(currentIndex);
    }
}
```

This example covers moving along a single line segment, which is admittedly not very interesting. The next recipe, 3.6, addresses navigating along line segment networks.

Discussion

When incrementing or decrementing the LengthIndexedLine indices, ensure that you don't exceed the start or end point index values. I.e., truncate them to the line start or end point value when a increment or decrement exceeds that value. The highlighted line 32 demonstrates checking for stepping off the

line segment. If you want to move the agent on to a connecting line segment, then take the truncated balance and apply that to the index for the next LengthIndexedLine.

sim.app.geo.campusworld, sim.app.geo.gridlock, and sim.app.geo.networkworld are GeoMason examples of agents moving along a line.

3.5 Creating a Network Graph from Lines

Problem

You have a lines describing a network, such as roads, that you would like to have as a single network, or graph.

Solution

GeoMason supplies a class, GeomPlanarGraph, that can be populated from a GeomVectorField.

```
GeomVectorField vectorField = new GeomVectorField(WIDTH,HEIGHT);
// ...
GeomPlanarGraph graph = new GeomPlanarGraph();
graph.createFromGeomField(vectorField);
```

Discussion

Once you have such a graph, it then becomes easier to compute shortest paths as in recipe 3.6.

3.6 Calculating the Shortest Path on a Network

Problem

You want to find the shortest path between two points in a network such as set of roads.

Solution

First, create a planar graph, as in recipe 3.5. Then use a shortest path algorithm, such as Dijkstra's[3] or A^* [4] to compute the shortest route for each agent.

Discussion

Computing shortest paths in a network is a well studied problem for which there exist a number of solutions, including solutions oriented towards transportation networks.[5]

Once the shortest path is computed you can use recipe 3.4 to move the agent along the route.

See Also

The GridLock demo creates a GeomPlanarGraph and an A^* algorithm to plot optimal commuter routes for agents. This demo can be found at sim.app.geo.gridlock.

3.7 Breaking Up Lines By Intersections

Problem

You want to ensure that lines are broken up by their intersections, which in JTS parlance is otherwise known as "noding".

[3] http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
[4] http://en.wikipedia.org/wiki/A*_search_algorithm
[5] http://en.wikipedia.org/wiki/Shortest_path_problem

Solution

Create a MultiLineString from the set of lines of interest and then computing their union(). The following is an example that uses this technique to split two lines that intersect in a 'T' formation into three line segments.

```
GeometryFactory geometryFactory = new GeometryFactory();
WKTReader reader = new WKTReader(geometryFactory);

ArrayList<LineString> rawlines = new ArrayList<LineString>();

rawlines.add((LineString) reader.read("LINESTRING (0 1, 1 1)"));
rawlines.add((LineString) reader.read("LINESTRING (1 0, 1 3)"));

MultiLineString newLines =
    new MultiLineString(rawlines.toArray(new LineString[0]), geometryFactory);

System.out.println("Before computing intersection:\n" + newLines);

Geometry brokenUpLines = newLines.union();

System.out.println("Broken up by intersection:\n" + brokenUpLines);
```

The output from this code shows the two lines are now broken into three lines split by the original two lines' intersection:

```
Before computing intersections:
MULTILINESTRING ((0 1, 1 1), (1 0, 1 3))
Broken up by intersection:
MULTILINESTRING ((0 1, 1 1), (1 0, 1 1), (1 1, 1 3))
```

JTS offers a third approach to computing line intersections. You can use a "Noder" that explicitly calculates the intersections between all the given geometry as shown in the following example.

```
GeometryFactory geometryFactory = new GeometryFactory();
WKTReader reader = new WKTReader(geometryFactory);

MultiLineString multiLine =
    (MultiLineString) reader.read("MULTILINESTRING ((0 1, 1 1),(1 3, 1 0))");

System.out.println("Before computing intersections: " + multiLine);

List lines = SegmentStringUtil.extractSegmentStrings(multiLine);

System.out.println("Converted to segmented lines: " + lines);

RobustLineIntersector RLI = new RobustLineIntersector();
IntersectionAdder SI = new IntersectionAdder(RLI);

```

```
16 MCIndexNoder noder = new MCIndexNoder();
17 noder.setSegmentIntersector(SI);
18
19 noder.computeNodes(lines);
20
21 System.out.println("Noded Noded Substrings: " + noder.getNodedSubstrings());
```

This code produces the following output:

```
Before computing intersections: MULTILINESTRING ((0 1, 1 1), (1 3, 1 0))
Converted to segmented lines: [LINESTRING (0.0 1.0, 1.0 1.0), LINESTRING (1.0 3.0, 1.0 0.0)]
Noded Substrings: [LINESTRING (0.0 1.0, 1.0 1.0), LINESTRING (1.0 3.0, 1.0 1.0),
   LINESTRING (1.0 1.0, 1.0 0.0)]}
```

Note that the noder returns a list of NodedSegmentStrings, which may be inconvenient.

Discussion

This is usually a prerequsite for computing line intersections, which is described in recipe 3.8. Though the examples are comprised entirely of JTS objects, recall that you can access JTS Geometry objects via the MasonGeometry.geometry data member.

3.8 Computing Line Intersections

Problem

You want to find all the intersections for a set of lines. For example, you may want to locate all road junctions.

Solution

Use the JTS Topology Suite Geometry method intersection() to compute the intersecting Point. The following code gives an example of using intersection().

```
GeometryFactory geometryFactory = new GeometryFactory();
WKTReader reader = new WKTReader(geometryFactory);

LineString firstLine =
    (LineString) reader.read("LINESTRING (0 10, 10 10)");
LineString secondLine =
    (LineString) reader.read("LINESTRING (5 20, 5 0)");

Geometry intersection = firstLine.intersection(secondLine);

System.out.println("Intersection of " + firstLine + " and " + secondLine + " is " + intersection);
```

The output of this code is "Intersection of LINESTRING (0 10, 10 10) and LINESTRING (5 20, 5 0) is POINT (5 10)", as you would expect.

However, generally we're more interested in finding the intersections for a set of lines. Unfortnately we're not guaranteed that our data is *coplanar*; that is, that when lines cross one another that they have an explicit vertex at that intersection. Before we can compute the points for all the intersections,

we must first ensure all the data is coplanar. Fortunately JTS has ways of breaking up lines by their intersections. This can be done by Once we're guaranteed that the lines are coplanar, we then need to find the points that define the line network intersections. You can do this getting the line start and end point coordinates, and then noting which of those occur more than once; those would be the points corresponding to line intersections.

The following code takes lines noded from the previous code example that uses the MCIndexNoder and prints the intersection value.

```
// contains noded line strings
Collection nodedLines = noder.getNodedSubstrings();

Set<Coordinate> intersectionVertices = new HashSet<Coordinate>();
Set<Coordinate> visitedCoordinates = new HashSet<Coordinate>();

for (Object line : nodedLines.toArray()) {
    NodedSegmentString curLine = (NodedSegmentString) line;

    Coordinate [] coordinates = curLine.getCoordinates();

    // Start point
    if (! visitedCoordinates.contains(coordinates[0])) {
        visitedCoordinates.add(coordinates[0]);
    } else {
        intersectionVertices.add(coordinates[0]);
    }

    // End point
    if (! visitedCoordinates.contains(coordinates[coordinates.length - 1])) {
        visitedCoordinates.add(coordinates[coordinates.length - 1]);
    } else {
        intersectionVertices.add(coordinates[coordinates.length - 1]);
    }
}

for (Coordinate c : intersectionVertices.toArray(new Coordinate[0])) {
    System.out.println(c);
}
```

The output is (1.0, 1.0, NaN), which is what we'd expect. (The NaN is for the Z coordinate, which isn't used in GeoMason.)

JTS does provide a simple one off means of getting line segment intersections. The following code demonstrates this; brokenUpLines is the noded JTS MultiLineString from the example that used union().[6]

```
System.out.println("Noded Substrings: " + brokenUpLines);

Geometry intPoints = (new BoundaryOp(brokenUpLines,
```

[6]Thanks to Martin Davis, the JTS author, for this example.

```
    BoundaryNodeRule.MULTIVALENT_ENDPOINT_BOUNDARY_RULE)).getBoundary();

System.out.println("Intersection: " + intPoints);
```

The output from this is:

```
Noded Substrings: MULTILINESTRING ((0 1, 1 1), (1 3, 1 1), (1 1, 1 0))
Intersection: POINT (1 1)
```

Discussion

Note that the JTS Geometry can be directly accessed from wrapping MasonGeometry objects as the geometry member.

3.9 Calculating Agent Information from Underlying Grid Data

Problem

You wish to to compute something based on values found in a grid an agent occupies. For example, a grid may represent a hectare and an agent may want to consume certain amount of vegetation and water found there.

Solution

Assume that you have an agent in a GeomVectorField and a corresponding GeomGridField layer that has information that agent needs to get or modify. First, you need to determine the grid cell coordinate for the agent. Then you can use that coordinate to fetch the grid cell contents.

```
GeomVectorField agent = new GeomVectorField(WIDTH,HEIGHT);
GeomGridField gridField = new GeomGridField();

// ... assume agent and gridField populated at this point

// assume first object in agent layer corresponds to agent of interest
Point location = agent.getGeometries().objs[0];

int x = gridField.toX(location);
int y = gridField.toY(location);

// assume grid field is of integers
int value = ((IntGrid2D)gridField.getGrid()).get(x,y);
```

Discussion

Naturally the above code example can be generalized to cover the other MASON 2D grid types.

This assumes that the grid and vector layers have the same underlying coordinate reference system and have had their minimum bounding rectangles synchronized as per recipe 2.1.5.

In a way, this is the inverse of recipe 3.1 — that is, instead of going from a grid layer to a vector, we're going from a vector to the grid.

3.10 Having an Agent Follow a Gradient

Problem

You have slope information that you would like an agent to follow.

Solution

Let's assume that you have a grid file containing elevation postings; i.e., essentially a matrix of floating point values that denote a height. An agent can be an arbitrary point associated with the elevation grid that's initialized at some random location. For each time step, the agent can get the height from its corresponding grid cell, and then find the adjacent cell with the smallest height and then move to that cell. The code below is a simple implementation of this approach.

```
public class Gradient extends SimState
{
    // field of elevation data
    GeomGridField elevation = new GeomGridField();

    // denotes agent's location; randomly set in start()
    int agentX;
    int agentY;

    public Gradient(long seed)
    {
        super(seed);

        InputStream elevationInputStream = null;

        try
        {
            elevationInputStream = new FileInputStream("elevations.asc");
        } catch (FileNotFoundException ex)
        {
            Logger.getLogger(Gradient.class.getName()).log(Level.SEVERE, null, ex);
        }

        ArcInfoASCGridImporter.read(elevationInputStream, GridDataType.DOUBLE, elevation);
    }


    @Override
    public void start()
    {
        super.start();

        // randomly place agent
        agentX = random.nextInt(elevation.getGridWidth());
        agentY = random.nextInt(elevation.getGridHeight());

        schedule.scheduleRepeating(new Mover());
    }
```

```
    // Steppable that moves the agent down slope
    private class Mover implements Steppable
    {
        public Mover()
        {}

        // heights of nearby cells
        DoubleBag heights = new DoubleBag();

        // Xs of nearby cells
        IntBag Xs = new IntBag();

        // Ys of nearby cells
        IntBag Ys = new IntBag();


        @Override
        public void step(SimState state)
        {
            Gradient gradient = (Gradient) state;

            // first get the adjacent eight cells and current cell

            heights = ((DoubleGrid2D)
            elevation.getGrid()).getNeighborsHamiltonianDistance(gradient.agentX,
                                                                 gradient.agentY,
                                                                 1, // distance
                                                                 false, // not torodial
                                                                 heights, Xs, Ys);

            // now find the cell with the lowest height

            int indexOfMin = 0; // index of heights with lowest elevation

            double currentMin = heights.get(0); // current lowest elevation

            for (int i = 1; i < heights.size(); i++) {
                if (heights.get(i) < currentMin) {
                  indexOfMin = i;
                  currentMin = heights.get(i);
                }
            }

            // Now set agent coordinate to minimum
            gradient.agentX = Xs.get(indexOfMin);
            gradient.agentY = Ys.get(indexOfMin);
        }
    }
}
```

Note that as an optimization the DoubleBag and IntBag objects in Mover in lines 48, 51, and 54 are declared outside of step() so that they are re-used instead of being recreated with each simulation step.

Discussion

The Water World demo, found as sim.app.geo.waterworld, gives a more elaborate example of how to follow a gradient. This demo uses elevation data for Crater Lake, Oregon, in a water flow model. In this model, rain drops are randomly dropped onto the grid; some of the rain drops fill the cadera while the rest runs down the gradients. Additionally, once the water level exceeds the height of the rim, the crater water then flows down streams from the crater.

The silly peds model is another gradient descent-based model. It overlays a building floor plan onto a grid of heights; the agents follow the height gradient out of the building. This demo can be found as sim.app.geo.sillypeds.

Fig. 3.1 shows screenshots of both of these demos.

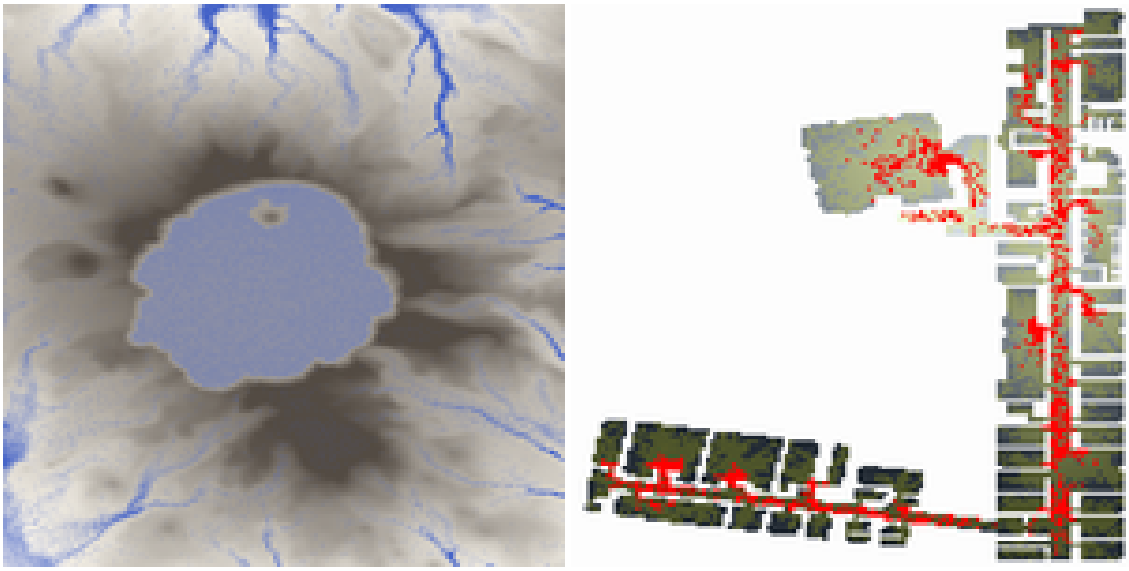


Figure 3.1 Screenshots of WaterWorld and SillyPeds models.

Chapter 4

Displaying Geospatial Data

This chapter gives recipes for displaying GeoMason fields in MASON.

4.1 Displaying a GeomVectorField

Problem

You want to show the contents of a GeomVectorField in a MASON display.

Solution

Create a GeomVectorFieldPortrayal in the MASON GUIState subclass, associate it with its corresponding GeomVectorField, set up an appropriate MASON or GeoMason field portrayal, and attach it to a MASON Display2D object.

```
public class MyMasonGUI extends GUIState
{
    private Display2D display;
    private JFrame displayFrame;

    // ... other variable declarations

    private GeomVectorFieldPortrayal myPortrayal = new GeomVectorFieldPortrayal();

    @Override
    public void init(Controller controller)
    {
        super.init(controller);

        display = new Display2D(ARBITRARY_WIDTH, ARBITRARY_HEIGHT, this);

        display.attach(myPortrayal, "My Vector Layer");

        displayFrame = display.createFrame();
        controller.registerFrame(displayFrame);
        displayFrame.setVisible(true);
    }
```

```
    @Override
    public void start()
    {
        super.start();
        setupPortrayals();
    }

    private void setupPortrayals()
    {
        MyState world = (MyState)state;

        myPortrayal.setField(world.vectorLayer);
        myPortrayal.setPortrayalForAll(new GeomPortrayal(Color.CYAN, true));

        display.reset();
        display.setBackdrop(Color.WHITE);

        display.repaint();
    }

    // ... other code
}
```

Discussion

Line 36 creates a portrayal that draws all the lines in cyan; the optional second parameter, which is set to true, indicates that polygons should be filled.

Another optional parameter, which is not shown in the above example, is for the scale to which geometry should be drawn. Care should be given when setting this parameter as the scale is in the units of the underling coordinate reference system. For example, this means that if you are using UTM all geometry will be in scaled to a be in meters, or in degrees if you are using lat/lon coordinates. If you are not careful, you can have scale values that either render geometry so small so as to not be visible, or so large as to obscure the entire display. Note that by default the scale value is 1, which may be too small if the simulation area is quite large and the units are in meters; conversely it may be too large if the units are in degrees. Some experimentation for the correct scale value may be necessary. Recipe 5.1 addresses problems associated with bad GeomPortrayal scale factor values.

Note that legacy MASON potrayals can be used. E.g., if you have agents represented as points in a GeomVectorField you might decide to use a MASON RectanglePortrayal2D instead of GeomPortrayal.

4.2 Displaying a GeomGridField

Problem

You want to show the contents of a GeomGridField in a MASON display

Solution

Set up an appropriate field portrayal for the wrapped Grid2D object found inside the GeomGridField .

```
public class MyMasonGUI extends GUIState
{
```

```
    private Display2D display;
    private JFrame displayFrame;

    // ... other variable declarations

    private FastValueGridPortrayal2D myPortrayal = new FastValueGridPortrayal2D();

    @Override
    public void init(Controller controller)
    {
        super.init(controller);

        display = new Display2D(ARBITRARY_WIDTH, ARBITRARY_HEIGHT, this);

        display.attach(myPortrayal, "My Grid Layer");

        displayFrame = display.createFrame();
        controller.registerFrame(displayFrame);
        displayFrame.setVisible(true);
    }

    @Override
    public void start()
    {
        super.start();
        setupPortrayals();
    }

    private void setupPortrayals()
    {
        MyState world = (MyState)state;

        myPortrayal.setField(world.gridLayer.getGrid());
        myPortrayal.setMap(new SimpleColorMap(0, 1, Color.black, Color.white));

        display.reset();
        display.setBackdrop(Color.WHITE);

        display.repaint();
    }

    // ... other code
}
```

Discussion

A GeomGridField is effectively a wrapper round a MASON Grid2D object, which means you can use the display techniques for Grid2D objects. Just use the GeomGridField.getGrid() method to fetch the underlying IntGrid2D or DoubleGrid2D object, as appropriate for the data type you specified when you read the grid data. (See recipe 2.1.4 for how to specify the grid data representation.)

The above code sample is for reading a grid layer that's presumably comprised of just ones and

zeroes. So a FastValueGridPortrayal2D will suffice to render that layer, as shown in line 8, along with an associated SimpleColorMap to show the zeros in black and the ones in white, as seen in line 36. The grid is attached to the field as in non-GeoMason MASON simulations; line 35 shows the getGrid() call necessary to get at the underyling MASON IntGrid2D.

4.3 Displaying Boundary Lines Over a Grid Field

Problem

You want to overlay political boundaries on grid data.

Solution

Load the vector and grid layers as per recipe 2.1.5. Then set up the portrayals such that the vector layer is rendered on top of the raster layer.

Reading the layers

```
GeomVectorField boundaries = new GeomVectorField(WIDTH,HEIGHT);
GeomGridField gridField = new GeomGridField();

try {
   ShapeFileImporter.read("file:boundaries.shp", boundaries);

   InputStream inputStream = new FileInputStream("grid.asc");
   ArcInfoASCGridImporter.read(inputStream, GridDataType.INTEGER, gridField);
} catch (FileNotFoundException ex)
{  /* handle exception */  }

Envelope globalMBR = boundaries.getMBR();

globalMBR.expandToInclude(gridField.getMBR());

boundaries.setMBR(globalMBR);
gridField.setMBR(globalMBR);
```

Displaying boundaries over the grid

```
public class MyMasonGUI extends GUIState
{
    private Display2D display;
    private JFrame displayFrame;

    // ... other variable declarations

    private FastValueGridPortrayal2D gridPortrayal = new FastValueGridPortrayal2D();
    private GeomVectorFieldPortrayal boundariesPortrayal = new GeomVectorFieldPortrayal();

    @Override
    public void init(Controller controller)
    {
        super.init(controller);
```

```
        display = new Display2D(ARBITRARY_WIDTH, ARBITRARY_HEIGHT, this);

        display.attach(gridPortrayal, "My Grid Layer");
        display.attach(boundariesPortrayal, "Political Boundaries");

        displayFrame = display.createFrame();
        controller.registerFrame(displayFrame);
        displayFrame.setVisible(true);
    }

    @Override
    public void start()
    {
        super.start();
        setupPortrayals();
    }

    private void setupPortrayals()
    {
        MyState world = (MyState)state;

        myPortrayal.setField(world.gridField.getGrid());
        myPortrayal.setMap(new SimpleColorMap(0, 1, Color.black, Color.white));

        myPortrayal.setField(world.boundaries);
        myPortrayal.setPortrayalForAll(new GeomPortrayal(Color.GRAY, true));

        display.reset();
        display.setBackdrop(Color.WHITE);

        display.repaint();
    }

    // ... other code
}
```

Discussion

From the perspective of the GUIState, the vector layer of boundaries are just yet another field portrayal rendered on top of another showing grid data. If the minimum bounding rectangles (MBR) of the two fields were properly aligned — and that they have the same coordinate reference system! — then the boundaries should match up with the grid layer geospatial features.

You can load additional grid layers before the vector layer, and you can toggle their visibility in the MASON Display2D window. It's similarly possible to load multiple boundaries, though admittedly displaying them all at once may be confusing; however, this confusion may be somewhat mitigated with good choices for line colors and widths. Just ensure that the vector layers always are attach()'d *after* all the grid layers because otherwise the grid layers will obscure the boundaries.

4.4 Displaying a Dynamic Choropleth Map

Problem

You want to create a dynamic choropleth map with colors changing based on agent behavior.

Solution

Have one GeomVectorField for agents and another containing political boundary polygons. Create a MasonGeometry subclass to be used for the political boundary polygons that will count the agents that are within its borders. Create a corresponding GeomPortrayal subclass that scales the color according to the agent count reported by that MasonGeometry subclass.

The following code shows how to use the special MasonGeometry subclass when reading the shape file. The highlighted lines shows that the class object for the counting wrapper MasonGeometry is passed in to read() so that that class is used instead of the default MasonGeometry.

In Your SimState

```
public class MyWorld extends SimState {

    // used in GUIState to set up ColorMap
    public static int NUM_AGENTS = 20;

    public GeomVectorField borders = new GeomVectorField(WIDTH,HEIGHT);

    // public static so CountingMasonGeometry can access
    public static GeomVectorField agents = new GeomVectorField(WIDTH,HEIGHT);

    public MyWorld(long seed) {
        super(seed);

        URL politicalBoundaries = MyWorld.class.getResource("borders.shp");

        try {
            ShapeFileImporter.read(politicalBoundaries, border, CountingGeomWrapper.class);
        } catch (FileNotFoundException ex) {
            // handle exception
        }
    }

    // ... other code
}
```

This is the class that is a wrapper round MasonGeometry. It just counts the number of agents that the given geometry "covers"; i.e., the agents that are within the boundaries of this object.

MasonGeometry subclass that counts agents

```
public class CountingGeomWrapper extends MasonGeometry {
    public CountingGeomWrapper()  {
        super();
    }

    public int numAgentsInGeometry()  {
```

```
        Bag coveredAgents = MyWorld.agents.getCoveredObjects(this);
        return coveredAgents.numObjs;
    }
}
```

This portrayal doesn't do anything special. It just takes the number of agents from the special counting MasonGeometry object to lookup the appropriate color in the color map, and then renders the overall geometry using that color.

Portrayal that renders choropleth

```
public class WorldPortrayal extends GeomPortrayal {

    SimpleColorMap colorMap = null;

    public WorldPortrayal(SimpleColorMap map) {
        super(true);
        colorMap = map;
    }

    @Override
    public void draw(Object object, Graphics2D graphics, DrawInfo2D info) {
       CountingGeomWrapper gm = (CountingGeomWrapper)object;
       paint = colorMap.getColor(gm.numAgentsInGeometry());
       super.draw(object, graphics, info);
    }
}
```

And, finally, in the GUIState class, the appropriate GeomVectorFieldPortrayal is set up to use our special field portrayal with a color map based on the total number of agents in the simulation.

Set up portrayal in GUIState

```
    GeomVectorFieldPortrayal borderPortrayal = new GeomVectorFieldPortrayal();
    GeomVectorFieldPortrayal agentPortrayal = new GeomVectorFieldPortrayal();

    private void setupPortrayals()   {
        MyWorld world = (MyWorld) state;

        agentPortrayal.setField(MyWorld.agents);
        agentPortrayal.setPortrayalForAll(new OvalPortrayal2D(Color.RED, 6.0));

        countyPortrayal.setField(world.borders);
        countyPortrayal.setPortrayalForAll(new MyWorldPortrayal(
            new SimpleColorMap(0.0, MyWorld.NUM_AGENTS, Color.WHITE, Color.BLUE)));

        display.reset();
        display.setBackdrop(Color.WHITE);
        display.repaint();
    }
```

Discussion

Fig. 4.1 shows a snapshot of the "ColorWorld" demo the shows a choropleth of the Fairfax County voting districts; the blue shading is in proportion to the number of agents that happen to be within a given voting district at each time step. This demo can be found as sim.app.geo.colorworld.

Again, this above code exemplifies the approach that the "ColorWorld" demo uses — almost certainly there are other equally viable approaches to implementing dynamic choropleths in GeoMason.

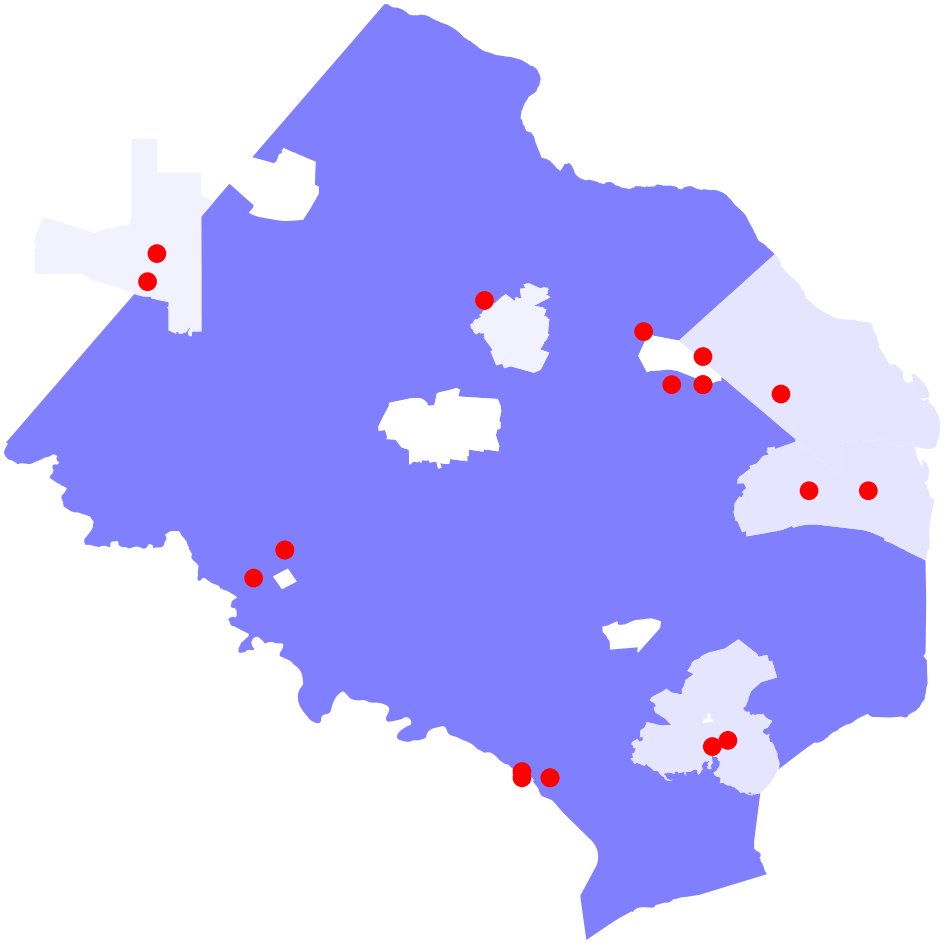


Figure 4.1 Snapshot of "Color World" demo showing polygon shading in proportion to number of agents.

4.5 Displaying Raster Overlays

Problem

You have raster overlays you wish to render on top of other display layers. For example, you may wish to overlay a heatmap representing population size, however you don't want the heatmap to hide details beneath it.

Solution

You can use alpha transparency when creating the grid field portrayal.

```
// Map population of grid cell from [0,25000] to shades of red from [0,255] with
// alpha transparency of 100/255.
myPortrayal.setMap(new SimpleColorMap(0, 25000, new Color(0,0,0,100), new Color(255,0,0,100)));
```

Discussion

Some experimentation may be necessary to determine the optimal alpha transparency level.

Chapter 5

Common Problems

5.1 Display All One Color

Problem

Rendering a GeomField just shows one solid color.

Solution

It is likely that the scale factor you are using for your GeomPortrayal is too large. Try using a scale factor that makes sense for the underlying coordinate reference system. E.g., if you are using UTM, the units will be in meters; calculate the total display area in meters and scale the agents accordingly.

See also recipe 4.1.

Discussion

Fig. 5.1 shows the GeoMason Grid Lock demo. The left subfigure shows the demo with the scale factor properly tailored to the scale of the underlying coordinate reference system. The right subfigure shows what happens if you go with the default scale factor of 1.0: the agents now obscure most of the display. You can hopefully see pathological variants where the agents cover the entire display in a single color. In this case we at least have a clue with part of the Virginia road network peeping out from behind the rendered agents.

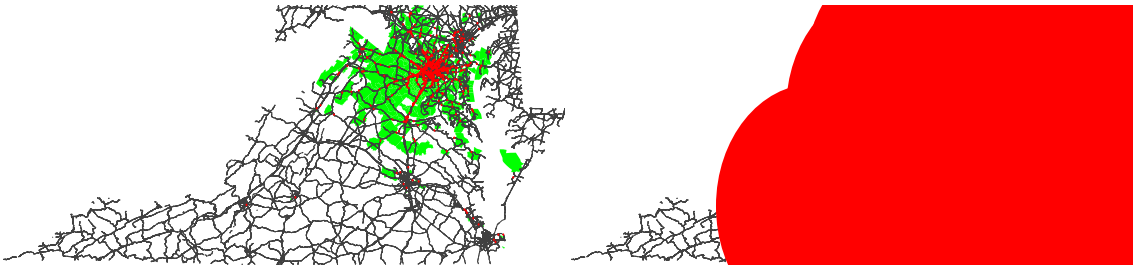


Figure 5.1 The left subfigure shows proper scaling of the agents; the right shows what happens when the default of 1.0 is used.

5.2 Layers Do Not Align

Problem

The data between layers does not match.

Solution

You probably did not align the minimum bonding rectangles between all the GeomField layers as directed by recipe 2.1.2.

Discussion

Fig. 5.2 shows what happens when the minimum bounding rectangles (MBR) between layers are not synchronized. Both figures show the GeoMason Campus World demo. The one on the left shows the demo with the MBRs properly synchronized. The figure on the right shows the same demo, but this time all the code that is responsible for synchronizing the MBRs has been removed. Note that the buildings are no longer inside the roads and that the agents are not even visible. More specifically, given that the underlying coordinate reference system for this demo is UTM, the units are in meters. The unmodified building MBR is a rectangle with defining corner coordinates (1182163, 6986772), (1182361, 6988786) and the corresponding default agent MBR is (10, 10), (10, 10), which would explain why the agents are not visible.

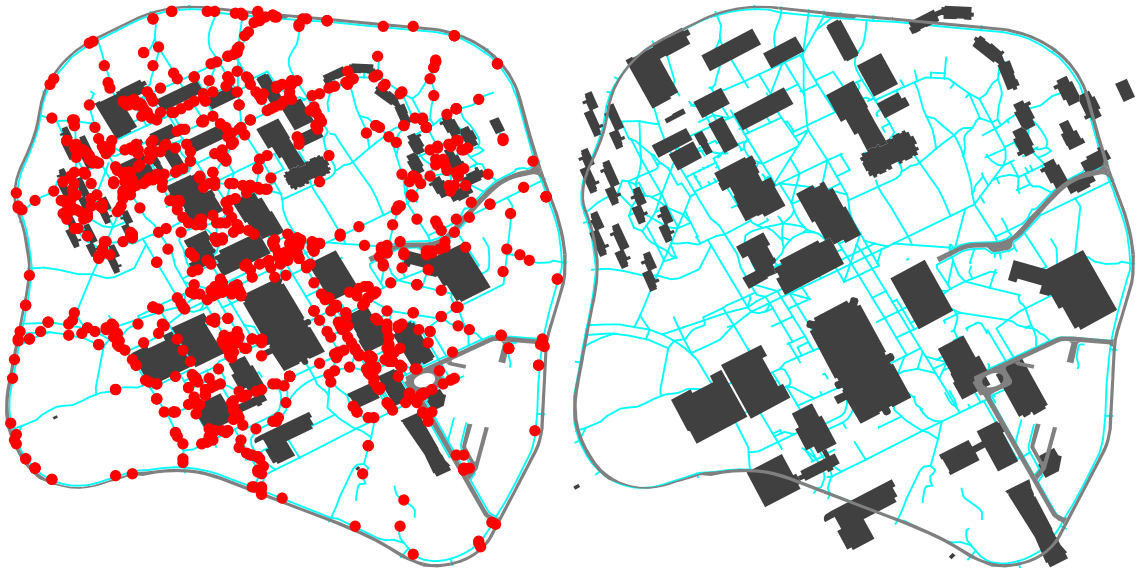


Figure 5.2 Shows proper MBR alignment on the left, and on the right what happens when the MBRs are not set at all

Acknowledgements

Thanks to the support of the Office of Naval Research (ONR) under a Multidisciplinary University Research Initiative (MURI) Grant No. N00014-08-1-0921, the Joint Improvised Explosive Device Defeat Ofce (JIEDDO) J-9 Division and the Ofce of Naval Research (ONR) under Government Contract number N00014-09-C-0419, and also NSF grant 0916870 for supporting this work. I would also like to thank the entire MASON development team and the GMU Center for Social Complexity without whom this work would not have been possible. I also would like to give thanks to Andrew Crooks for valuable feedback and for providing most of the GeoMason demos that originated as student class projects from his classes. I would also like to thank Martin Davis for writing JTS Topology Suite and for providing valuable help and feedback in its use. And, finally, I would like to thank Keith Sullivan for his contribution to GeoMason; he did yeoman service in writing the initial incarnations of the native GeoMason shape file reader and writer classes as well as getting GeoMason portrayals up to speed.

Index