Project: *AORS: Visualization Module – rendering algorithms*
Date: *November, 2009*
Document version: *1.0*
Author: *Mircea Diaconescu*

AORS: Visualization

Render Engine Description

Table of Contents

A: Introduction

The purpose of this document is to describe how the rendering engine must perform the actions in order to get the simulation drawn. The following algorithms describe the steps that have to be performed in order to provide e simulation visualization.

B: Render Algorithm

```
// define the variables used by this algorithm
p0: DEFINE: renderEngine, drawBuffer, drawMethod, spaceModel,
            spaceView, initialState, simulationSteps,
            totalSimulationStep, currentDrawStep, entitiesList.

// create a render engine instance
p1: renderEngine = RenderEngine.createRenderEngine();
```

Project: ***AORS: Visualization Module – rendering algorithms***
Date: ***November, 2009***
Document version: ***1.0***
Author: ***Mircea Diaconescu***

```
// create the drawBuffer. Please note that the buffer is filled
// with simulation steps by the simulation engine via event notifiers
// procedures. The draw engine just look in the buffer and draw the
// next step if there is one in the buffer. The draw buffer is a QUEUE
// implementation FIFO (First Input First Output)
p5: drawBuffer = RenderEngine.createDrawBuffer();

// we have to choose if we draw based on the log or we draw
// real time simulation steps
p7: drawMethod =  REAL_TIME_SIMULATION_BASED;
       OR
    drawMethod = LOG_BASED;

// create spaceModel object from XML file
p10: spaceModel = XMLReader.readSpaceModel();

// create spaceView object from XML file
p20: spaceView = XMLReader.readSpaceView();

// create initialState object from XML file
p25: intialState = XMLReader.readInitialState();

// draw the space based on the space model and space view
// this method si described in algorithm C - Space Drawing Algorithm
p30: renderEngine.drawSpace(spaceModel, spaceView);

// extract the entities defined in the initial state
p33:  entitiesList = initialState.getEntitiesList();

// draw the initial state. This method draws the state of
// the simulation before start.
// The algorithm: D - Entities Drawing Algorithm, describes this method
p35: renderEngine.drawStep(entitiesList, spaceModel.isDiscrete());

p40: simulationStep = 0;
     totalSimulationSteps = Simulation.getTotalSteps();

// the loop where the simulation steps are drawn
p50: while (simulationStep <= totalSimulationSteps)

       p50.0:  if(drawMethod == REAL_TIME_SIMULATION_BASED)

                  // get the next step from the buffer
                  currentDrawStep =  drawBuffer.getStep(0);

                  // delete this step from the buffer
                  drawBuffer.deleteStep(0);

               else if (drawMethod == LOG_BASED)

                       // get the next step from the log file
                       currentDrawStep = XMLReader.getNextStep()
```

```
            // the step may be null in the case of real time drawing
            p50.5:  if (currentStep == null) then goto p50.0;

            // draw the step only it has some changes, otherwise  there is 
            // no need to spend time to redraw the space and everything
            p50.10:  if(currentDrawStep.containsDrawChanges())

                        // update the list of entities with the changes
                        // defined in the current simulation step
                        renderEngine.updateEntitiesList(currentDrawStep);

                        // clear the draw area
                        renderEngine.clearDrawArea();

                        // draw the space 
                        renderEngine.drawSpace(spaceModel, spaceView);

                        // draw the entities
                        renderEngine.drawStep(entitiesList, spaceMode.isDiscrete());

            // increase the numbar of steps that were already
            // drawn by the rendering engine. This is increased
            // also if the current step has no changes inside
            p50.30:   simulationStep = simulationStep + 1;

    p60:  end while

    // reset the rendering engine. This means that all the structures, 
    // lists and other objects that may in the memory have to be cleaned 
    // freeing up the memory. Also, any threads that my be used in the
    // render engine have to be stopped.
    p70: renderEngine.reset();
```

C: Space Drawing Algorithm – RenderEngine.drawSpace

```
    // the parameters used by the algorithm
    p0: PARAMETERS: spaceModel, spaceView.

    P10: if(spaceModel.getDimensions() == regularExpresion(ONE_PLUS))
            renderSpace1D(spaceModel, spaceView);

    P20: if(spaceModel.getDimensions() == TWO)
            renderSpace2D(spaceModel, spaceView);

    P30: if(spaceModel.getDimensions() == THREE)
            renderSpace3D(spaceModel, spaceView);
```

Project: ***AORS: Visualization Module – rendering algorithms***
Date: ***November, 2009***
Document version: ***1.0***
Author: ***Mircea Diaconescu***

C1: Space 1D – RenderEngine.renderSpace1D

```
// define the input parameters of the algorithm
p0: PARAMETERS: spaceModel, spaceView.

// define the variables used by this algorithm
p5: DEFINE: viewportWidth, viewportHeight, spaceMaxX, spaceMaxY,
            spaceType, ratioWidth, ratioHeight, cellWidth, cellHeight,
            lineWidth, lineHeight.

// get the width and height of the viewable area of the visualization.
// With other words, these are the dimensions of the area where the user sees
// the visualization of the simulation.
p10: viewportWidth = getViewportWidthFromSystem();
     viewportHeight = getViewportHeightFromSystem();

// set default space type
p15: spaceType = HORIZONTAL
           OR
      spaceType = VERTICAL
           OR
      spaceType = CIRCULAR

// define a non drawable margin
p17: margin = 25; spaceMaxY = 1;

// get the maxX and maxY values
p18: spaceMaxX = spaceModel.getMaxX();
     spaceMaxY = spaceModel.getMaxY();

p19: ratioWidth = viewportWidth/spaceMaxX;
      ratioHeight = viewportHeight/spaceMaxY;

// load the space type from the space view if defined
p20: if(spaceView.getType() != null)
        spaceType = spaceView.getType();

p30: if(spaceModel.isDiscrete())

      p30.0: if(spaceType = HORIZONTAL)
                cellWidth = (viewportWidth -2*margin) / spaceMaxX;
                cellHeight = cellWidth;
      p30.10: if (spaceType = VERTICAL)
                 cellHeight = (viewportHeight - 2*margin) / spaceMaxX;
                 cellWidth = cellHeight;
```

```
          p30.20: for(i=1; i<spaceMaxX; i++)
                         // render horizontal or vertical cells based on spaceType value.
                         // this draw one after one the cells that describe the discrete
                         // space (on horizontal or vertical)
                         renderCell(i, spaceType, cellWidth, cellHeight);
          p30.30: end for

     p50: if(!spaceModel.isDiscrete())

          p50.0: if(spaceType == HORIZONTAL)

                    lineWidth = (viewportWidth -2*margin)*ratioWidth;

          p50.10: if (spaceType == VERTICAL)

                     lineWidth = (viewportHeight - 2*margin)*ratioHeight;

          p50.15: lineHeight = spaceView.getWidth();

          p50.20: for(i=1; i<spaceMaxY; i++)
                         // draw space line based on space type
                         renderLine(i, spaceType, lineWidth, lineHeight);
          p50.30: end for

     p60: if(spaceType = CIRCULAR)
             radiusX = (spaceMaxX * ratioWidth -2*margin) / 2;
             radiusY = (viewportHeight -2*margin) / 2;

             p60.10: for(i=1; i<=spaceMaxY; i++)

                          // draw space line based on space type
                          renderCircularSpace(i, lineHeight, radiusX, radiusY);
             p60.20: end for
```

C2: Space 2D – RenderEngine.renderSpace2D

```
// define the input parameters of the algorithm
p0: PARAMETERS: spaceModel, spaceView.

// define the variables used by this algorithm
p5: DEFINE: viewportWidth, viewportHeight.

// get the width and height of the viewable area of the visualization.
// With other words, these are the dimensions of the area where the user sees
// the visualization of the simulation.
p10: viewportWidth = getViewportWidthFromSystem();
     viewportHeight = getViewportHeightFromSystem();
```

```
// define a non drawable margin
p17: margin = 25;

// get the maxX and maxY values
p18: spaceMaxX = spaceModel.getMaxX();
     spaceMaxY = spaceModel.getMaxY();

// compute the aspect ratio. This is used to port the space size to
// the viewport size, and accorting with this the objects are scaled.
p19: ratioWidth = viewportWidth/spaceMaxX;
      ratioHeight = viewportHeight/spaceMaxY;

// load the space type from the space view if defined
p20: if(spaceView.getType() != null)
          spaceType = spaceView.getType();

p30: if(spaceModel.isDiscrete())
      // compute the cell size
      p30.0: cellWidth = (viewportWidth -2*margin) / spaceMaxX;
             cellHeight = cellWidth;

      p30.20: for(i=1; i<=spaceMaxY; i++)
                 // draw one by one the cell lines of the space
                 renderCellsLine(i, spaceMaxY, cellWidth, cellHeight);
      p30.30: end for
```

C3: Space 3D – RenderEngine.renderSpace3D

TODO.

D: Entities Drawing Algorithm – RenderEngine.drawStep

```
// define the input parameters of the algorithm
p0: PARAMETERS: entitiesList, discrete.

// define the variables used by this algorithm
p5: DEFINE: entity, ratioWidth, ratioHeight .

// compute the aspect ratio. This is used to port the space size to
// the viewport size, and accorting with this the objects are scaled.
p10: ratioWidth = viewportWidth/spaceMaxX;
      ratioHeight = viewportHeight/spaceMaxY;
```

```
// draw the entities from the list
p50: for(i=0; i<entitiesList.size(); i++)

     // get the next entity
     p50.10: entity = entitesList.get(i);

     // draw the entity. Notice that the entity contains the view object.
     // the discrete parameter is used to decide if the object is drawn
     // in a discrete space or in a continuous space. This will affect
     // the coordinate computation position. The drawShape method is dependent
     // on the graphic library (Canvas2D. O3D, etc).
     p50.20: drawShape(entity, discrete, ratioWidth, ratioHeight);

p60: end for
```