From MASON to D-MASON

ISISLab

Università degli Studi di Salerno

A simple porting: from Particles to DParticles

Package: sim.app.tutorial3 Particles

- **Particle:** agent of the simulation.
- **Particles:** extends SimState.
- **ParticlesWithUI:** visualization tool for simulations.

A simple porting: from Particles to DParticles

Package: dmason.sim.app.DParticles

- **RemoteParticle:** it provides an unique ID and field position for the agent
- **DParticle:** *Remote* agent of the simulation.
- **DParticles:** extends a *DistributedState*.
- **DParticlesWithUI:** visualization tool for simulations.

Particle

Particle implements Steppable

- **Constructor**
  - *Particle* implements *Steppable*
  - two parameters: *xdir*, *ydir*
- **step()**
  - *setObjectLocation()*: sets the agent in a given position of the field
  - *check for collision*

Particle Constructor

Listing 1: Contructor of Particle

```
...
  public Particle(int xdir, int ydir)
  {
   public boolean randomize = false;
   this.xdir = xdir;
   this.ydir = ydir;
  }
...
```

Particle *step()*

Listing 2: step() method

```
 ...
public void step(SimState state)
{
    ...
    if (randomize)
    {
        xdir = tut.random.nextInt(3) - 1;
        ydir = tut.random.nextInt(3) - 1;
        randomize = false;
    }
...
```

Particle *step()*

Listing 3: step() method

```
Int2D newloc = new Int2D(newx,newy);
tut.particles.setObjectLocation(this,newloc);

Bag p = tut.particles.getObjectsAtLocation(newloc);
if (p.numObjs > 1)
   {
    for(int x=0;x<p.numObjs;x++)
        ((Particle)(p.objs[x])).randomize = true;
    }
   }
```

DParticle

DParticle extends RemoteParticle

- **Constructor**
  - *empty constructor* for a future implementation of *clone()*
  - one parameter: a subclass of *DistributedState*
- **step()**
  - *setAvailableRandomLocation()*: assigns a random position to an agent in the distributed field
  - *setDistributedObjectLocation()*: sets the agent in a given position of the distributed field, in the next *snapshot* field
  - In the distributed environment agent first checks for collisions in the field at the previous step and then it sets its position

DParticle Constructor

Listing 4: Constructor of DParticle

```
public class DParticle
             extends RemoteParticle<Int2D>
{
 public int xdir;  // -1, 0, or 1
 public int ydir;  // -1, 0, or 1

 public DParticle(){ }

 public DParticle(DistributedState state)
 {
   super(state);
 }
```

DParticle step()

Listing 5: step() method

```
public void step(SimState state)
{
 DParticles tut = (DParticles)state;

 Int2D location = tut.particles.
                    getObjectLocation(this);

 Bag p = tut.particles.
                    getObjectsAtLocation(location);

 tut.trails.setDistributedObjectLocation
                  (1.0, location,state);
```

DParticle step()

Listing 6: step() method

```
if (p.numObjs > 1)
{
 xdir = tut.random.nextInt(3) - 1;
 ydir = tut.random.nextInt(3) - 1;
}
int newx = location.x + xdir;
int newy = location.y + ydir;
...
```

DParticle step()

Listing 7: step() method

```
if (newx < 0) { newx++; xdir = -xdir; }
else if (newx >= tut.trails.getWidth())
                           {newx--; xdir = -xdir; }
if (newy < 0) { newy++ ; ydir = -ydir; }
else if (newy >= tut.trails.getHeight())
                           {newy--; ydir = -ydir; }
Int2D newloc = new Int2D(newx,newy);
tut.particles.setDistributedObjectLocation
                                    (newloc, this, state);
  }
}
```

Particles

Particles extends SimState

- **Constructor**, as unique parameter, the random generator seed
- **Fields**
    - *SparseGrid2D* for the particles
    - *DoubleGrid2D* for the trails
- **scheduleRepeating()** for scheduling agents repeadetly
- **setAvailableRandomLocation()** for positioning agents in the field uniformly at random

Particles Constructor

Listing 8: Constructor

```
public class Particles extends SimState
{
  public DoubleGrid2D trails;
  public SparseGrid2D particles;
  ...
  public Particles(long seed)
  {
      super(seed);
  }
```

Particles start()

Listing 9: start() method

```
public void start()
{
...
 for(int i=0 ; i<numParticles ; i++)
 {
  p = new Particle(random.nextInt(3) - 1,
                       random.nextInt(3) - 1);
  schedule.scheduleRepeating(p);
  ...
  particles.setObjectLocation(p,new Int2D(x,y));
 }
}
```

DParticles

DParticles extends DistributedState

- **Constructor**, as parameter, *Object[]* array
- **Fields**
  - *DSparseGrid2D* for the particles
  - *DDoubleGrid2D* for the trails
- **scheduleOnce()** for scheduling agents at each step, because in the next step an agent could stay in another part of the field
- **setAvailableRandomLocation()** for positioning agents in the field uniformly at random
- **"getter"** for *DistributedState* and *DistributedField*

DParticles instances

Listing 10: Instances

```
public class DParticles
            extends DistributedState<Int2D> {

 private static boolean isToroidal=false;
 public DSparseGrid2D particles;
 public DDoubleGrid2D trails;
 public int gridWidth ;
 public int gridHeight;
```

DParticles Constructor

Listing 11: Constructor

```
public DParticles(Object[] params)
{
 super((Integer)params[2],(Integer)params[3],
 (Integer)params[4], (Integer)params[7],
 (Integer)params[8], (String)params[0],
 (String)params[1],(Integer)params[9], isToroidal,
new DistributedMultiSchedule<Int2D>());
 ip = params[0]+"";     port = params[1]+"";
 this.MODE=(Integer)params[9];
 gridWidth=(Integer)params[5];
 gridHeight=(Integer)params[6];
}
```

DParticles start()

Listing 12: start() method

```
public void start()
{
 super.start();
 try {
  trails = DDoubleGrid2DFactory.
     createDDoubleGrid2D(gridWidth,
     gridHeight, this, super.MAX_DISTANCE,
     TYPE.pos_i, TYPE.pos_j, super.NUMPEERS, MODE,
     0, false, ”trails”);
```

DParticles start()

Listing 13: start() method

```
  particles = DSparseGrid2DFactory.
    createDSparseGrid2d(gridWidth,
    gridHeight, this, super.MAX_DISTANCE,
    TYPE.pos_i, TYPE.pos_j,super.NUMPEERS,
    MODE, "particles");

  init_connection();
}catch (DMasonException e) { e.printStackTrace();}
```

DParticles start()

Listing 14: start() method

```
 DParticle p=new DParticle(this);
 while(particles.size() != super.NUMAGENTS)
{
 particles.setAvailableRandomLocation(p);
...
 if(particles.setDistributedObjectLocationForPeer
 (new Int2D(p.pos.getX(),p.pos.getY()), p, this))
 {
 schedule.scheduleOnce(schedule.getTime()+1.0,p);
...
 }
}
```

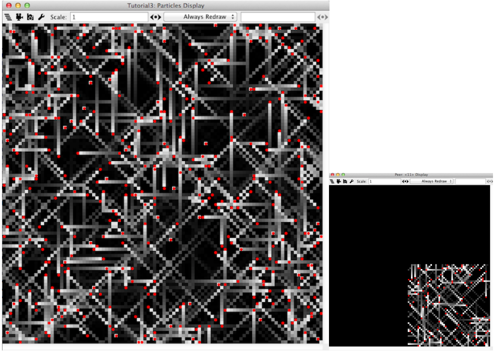
DParticles *getters*

Listing 15: getters methods

```
public DistributedField getField()
{ return particles; }

public SimState getState()
{ return this; }

public void addToField(RemoteAgent<Int2D> rm,Int2D
{ particles.setObjectLocation(rm, loc); }

public boolean setPortrayalForObject(Object o)
{ return false; }
}
```

ParticlesWithUI vs *DParticlesWithUI*

They both extend **GUIState**



Scale: 1
Always Redraw

Scale: 1
Always Redraw

Figure: Respectively *Particles* and a *DParticles*' parted field