NPC centralized control – on network server

SERVER SIDE (partial):

```
public class NetworkingServer
{ // same as before, plus an NPC control loop and NPC controller
 . . .
 private NPCcontroller npcCtrl;
 GameAIServerTCP tcpServer;

 public NetworkingServer(int id)              // constructor
 { startTime = System.nanoTime();
  lastUpdateTime = startTime;
  npcCtrl = new NPCcontroller();
  . . .
  // start networking TCP server (as before)
  . . .
  // start NPC control loop
  npcCtrl.setupNPCs();
  npcLoop();
 }

 public void npcLoop()                        // NPC control loop
 {  while (true)
  { long frameStartTime = System.nanoTime();
    float elapMilSecs = (frameStartTime-lastUpdateTime)/(1000000.0f);
    if (elapMilSecs >= 50.0f)
    { lastUpdateTime = frameStartTime;
      npcCtrl.updateNPCs();
      tcpServer.sendNPCinfo();
    }
    Thread.yield();
 } }
 // main() starts networking server as before
```

```
public class GameAIServerTCP extends GameConnectionServer<UUID>
{
 // game protocol as before, plus additional NPC protocol cases.  i.e.,
 // messages regarding NPC's sent to clients, such as:
 public void sendNPCinfo()    // informs clients of new NPC positions
 {  for (int i=0; i<npcCtrl.getNumOfNPCs(); i++)
   { try
    { String message = new String("mnpc," + Integer.toString(i));
      message += "," + (npcCtrl.getNPC(i)).getX();
      message += "," + (npcCtrl.getNPC(i)).getY();
      message += "," + (npcCtrl.getNPC(i)).getZ();
      sendPacketToAll(message);
    . . .
 // also additional cases for receiving messages about NPCs, such as:
 if(messageTokens[0].compareTo("needNPC") == 0)
 { . . . }
  if(messageTokens[0].compareTo("collide") == 0)
 { . . . }
}
```

```
public class NPC
{ double locX, locY, locZ;       // other state info goes here (FSM)
  public double getX() { return locX; }
  public double getY() { return locY; }
  public double getZ() { return locZ; }
  . . .
  public void updateLocation()  { . . . }
}
```

```
public class NPCcontroller
{
 private NPC[] NPClist = new NPC[5];
 . . .
 public void updateNPCs()
 { for (int i=0;  i<numNPCs;  i++)
   { NPClist[i].updateLocation();
 } }
 . . .
}
```

CLIENT SIDE (partial):

```
public class GhostNPC
{ private int id;
  private SceneNode node;
  private Entity entity;

  public GhostNPC(int id, Vector3 position)    // constructor
  {   this.id = id;
  }
  public void setPosition(Vector3 position)
  {   node.setLocatlPosition(position);
  }

  public void getPosition(Vector3 position)
  {   return node.getLocatlPosition();
  }
}
```

```
public class TestGameClient extends GameConnectionClient
{ // same as before, plus code to handle additional NPC messages
 . . .
 private Vector<GhostNPC> ghostNPCs;
 . . .
 private void createGhostNPC(int id, Vector3 position)
 {  GhostNPC newNPC = new GhostNPC(id, position);
    ghostNPCs.add(newNPC);
    game.addGhostNPCtoGameWorld(newNPC);
 }

 private void updateGhostNPC(int id, Vector3 position)
 {  ghostNPCs.get(id).setPosition(position);
 }

 // handle updates to NPC positions
 // format: (mnpc,npcID,x,y,z)
 if(messageTokens[0].compareTo("mnpc") == 0)
 { int ghostID = Integer.parseInt(messageTokens[1]);
   Vector3 ghostPosition = Vector3f.createFrom(
          Float.parseFloat(messageTokens[2]),
          Float.parseFloat(messageTokens[2]),
          Float.parseFloat(messageTokens[2]));
    updateGhostNPC(ghostID, ghostPosition);
 }

 public void askForNPCinfo()
 { try
   { sendPacket(new String("needNPC," + id.toString()));
   }
   catch (IOException e)
   { e.printStackTrace();
 } }
 . . .
```

```
public class MyGame extends VariableFrameRateGame
  now includes addGhostNPCtoGameWorld()
```