Networking for multiplayer games *(continued)*

TAGE client side protocol: *// UDP example protocol*

```
public class ProtocolClient extends GameConnectionClient
{   private MyGame game;
    private UUID id;
    private GhostManager ghostManager;
---------------------------------------------------------------
    public ProtocolClient(InetAddress remAddr, int remPort,
                ProtocolType pType, MyGame game) throws IOException
    {   super(remAddr, remPort, pType);
        this.game = game;
        this.id = UUID.randomUUID();
        ghostManager = game.getGhostManager();
    }
---------------------------------------------------------------
    @Override
    protected void processPacket(Object msg)
    {   String strMessage = (String)message;
        String[] messageTokens = strMessage.split(",");

        if(messageTokens.length > 0)
        {
            if(msgTokens[0].compareTo("join") == 0)        // receive "join"
            {   // format:  join, success  or  join, failure
                if(msgTokens[1].compareTo("success") == 0)
                {   game.setIsConnected(true);
                    sendCreateMessage(game.getPlayerPosition());
                }
                if(msgTokens[1].compareTo("failure") == 0)
                {   game.setIsConnected(false);
            }   }

            if(messageTokens[0].compareTo("bye") == 0)   // receive "bye"
            {   // format:  bye, remoteId
                UUID ghostID = UUID.fromString(messageTokens[1]);
                ghostManager.removeGhostAvatar(ghostID);
            }

            if  ((messageTokens[0].compareTo("dsfr") == 0 ) // receive "dsfr"
              || (messageTokens[0].compareTo("create")==0))
            {   // format:  create, remoteId, x,y,z  or  dsfr, remoteId, x,y,z
                UUID ghostID = UUID.fromString(messageTokens[1]);
                Vector3f ghostPosition = new Vector3f(
                        Float.parseFloat(messageTokens[2]),
                        Float.parseFloat(messageTokens[3]),
                        Float.parseFloat(messageTokens[4]));
                try
                {   ghostManager.createGhost(ghostID, ghostPosition);
                }   catch (IOException e)
                {   System.out.println("error creating ghost avatar");
            }   }

            if(messageTokens[0].compareTo("wsds") == 0)  // rec. "wants…"
            {  // etc…..  }
            if(messageTokens[0].compareTo("move") == 0)  // rec. "move…"
            {  // etc…..  }
    }   }
---------------------------------------------------------------
    public void sendJoinMessage()        // format:  join, localId
    {   try
        {   sendPacket(new String("join," + id.toString()));
        }   catch (IOException e)  { e.printStackTrace();
    }   }
---------------------------------------------------------------
    public void sendCreateMessage(Vector3 pos)
    {  // format:  (create, localId, x,y,z)
        try
        {   String message = new String("create," + id.toString());
            message += "," + pos.getX()+"," + pos.getY() + "," + pos.getZ();
            sendPacket(message);
        }
        catch (IOException e)  {  e.printStackTrace();
    }   }
```

also need code for:

- *public void **sendByeMessage**()*
- *public void **sendDetailsForMessage**(UUID remId, Vector3D pos)*
- *public void **sendMoveMessage**(Vector3D pos)*

```
public class GhostAvatar extends GameObject
{   private UUID id;

    public GhostAvatar(UUID id, ObjShape s, TextureImage t, Vector3f p)
    {   super(GameObject.root(), s, t);
        uuid = id;
        setPosition(p);;
    }
```

also need accessors and setters for id and position

```
public class GhostManager
{   private MyGame game;
    private Vector<GhostAvatar> ghostAvs = new Vector<GhostAvatar>();

    public GhostManager(VariableFrameRateGame vfrg)
    {   game = (MyGame)vfrg;
    }

    public void createGhost(UUID id, Vector3f p) throws IOException
    {   ObjShape s = game.getGhostShape();
        TextureImage t = game.getGhostTexture();
        GhostAvatar newAvatar = new GhostAvatar(id, s, t, p);
        Matrix4f initialScale = (new Matrix4f()).scaling(0.25f);
        newAvatar.setLocalScale(initialScale);
        ghostAvs.add(newAvatar);
    }

    public void removeGhostAvatar(UUID id)
    {   GhostAvatar ghostAv = findAvatar(id);
        if(ghostAvatar != null)
        {   game.getEngine().getSceneGraph().removeGameObject(ghostAv);
            ghostAvs.remove(ghostAv);
        }
        else
        {   System.out.println("unable to find ghost in list");
    }   }

    private GhostAvatar findAvatar(UUID id)
    {   GhostAvatar ghostAvatar;
        Iterator<GhostAvatar> it = ghostAvs.iterator();
        while(it.hasNext())
        {   ghostAvatar = it.next();
            if(ghostAvatar.getID().compareTo(id) == 0)
            {   return ghostAvatar;
        }   }
        return null;
    }

    public void updateGhostAvatar(UUID id, Vector3f position)
    {   GhostAvatar ghostAvatar = findAvatar(id);
        if (ghostAvatar != null) { ghostAvatar.setPosition(position); }
        else { System.out.println("unable to find ghost in list"); }
}   }
```

Game Application: (based on ex.02b)

```
. . .
import tage.networking.IGameConnection.ProtocolType;

public class MyGame extends VariableFrameRateGame
{  . . .
   private GhostManager gm;
   private String serverAddress;
   private int serverPort;
   private ProtocolType serverProtocol;
   private ProtocolClient protClient;
   private boolean isClientConnected = false;

   public MyGame(String serverAddress, int serverPort, String protocol)
   {  super();
      gm = new GhostManager(this);
      this.serverAddress = serverAddress;
      this.serverPort = serverPort;
      if (protocol.toUpperCase().compareTo("TCP") == 0)
         this.serverProtocol = ProtocolType.TCP;
      else
         this.serverProtocol = ProtocolType.UDP;
   }
```

```
   public static void main(String[] args)
   {  MyGame game =
            new MyGame(args[0], Integer.parseInt(args[1]), args[2]);
      // remainder as before
      . . .
   }
```

loadShapes(), loadTextures(), buildObjects(), initializeGame() as before

```
   private void setupNetworking()
   {  isClientConnected = false;
      try
      {  protClient = new ProtocolClient(InetAddress.
            getByName(serverAddress), serverPort, serverProtocol, this);
      }  catch (UnknownHostException e) { e.printStackTrace();
      }  catch (IOException e) { e.printStackTrace(); }
      if (protClient == null)
      {  System.out.println("missing protocol host"); }
      else
      {  // ask client protocol to send initial join message
         // to server, with a unique identifier for this client
         protClient.sendJoinMessage();
   }  }
```

```
@Override
protected void update(Engine engine)
   {  //  same as before, plus process any packets received from server
      . . . .
      processNetworking((float)elapsedTime)
   }
```

```
   protected void processNetworking(float elapsTime)
   {  // Process packets received by the client from the server
      if (protClient != null)
         protClient.processPackets();

   }
```

```
   public GameObject getAvatar() { return avatar; }
   public ObjShape getGhostShape() { return ghostS; }
   public TextureImage getGhostTexture() { return ghostT; }
   public GhostManager getGhostManager() { return gm; }
   public Engine getEngine() { return engine; }
```

```
   public Vector3 getPlayerPosition()
   {  return avatar.getWorldLocation(); }
```

```
   @Override
   public void keyPressed(KeyEvent e)
   {  switch (e.getKeyCode())
      {  case KeyEvent.VK_W:            // move avatar forward
         {  . . .                       // tell other players
            protClient.sendMoveMessage(avatar.getWorldLocation());
            break;
         }
         case KeyEvent.VK_D:
         {  . . .                       // turn avatar, tell other players
            break;
         }
      }
      super.keyPressed(e);
   }
```

```
   private class SendCloseConnectionPacketAction
                                        extends AbstractInputAction
   {  // for leaving the game... need to attach to an input device
      @Override
      public void performAction(float time, net.java.games.input.Event evt)
      {  if(protClient != null && isClientConnected == true)
         {  protClient.sendByeMessage();
   }  }  }
```

Avatar movement (in input action class):

```
package myGame;
import tage.*;
import tage.input.action.AbstractInputAction;
import net.java.games.input.Event;
import org.joml.*;

public class FwdAction extends AbstractInputAction
{  private MyGame game;
   private GameObject av;
   private Vector3f oldPosition, newPosition;
   private Vector4f fwdDir;
   private ProtocolClient protClient;

   public FwdAction(MyGame g, ProtocolClient p)
   {  game = g;
      protClient = p;
   }

   public void performAction(float time, Event e)
   {  av = game.getAvatar();
      oldPosition = av.getWorldLocation();
      fwdDir = new Vector4f(0f,0f,1f,1f);
      fwdDir.mul(av.getWorldRotation());
      fwdDir.mul(0.01f);
      newPosition = oldPosition.add(fwdDir.x(), fwdDir.y(), fwdDir.z());
      av.setLocalLocation(newPosition);
      protClient.sendMoveMessage(av.getWorldLocation());
}  }
```

(input handing shown using both methods)