

Networking for multiplayer games *(continued)*

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

public class ProtocolClient extends GameConnectionClient

```
{ private MyGame game;
  private UUID id;
  private Vector<GhostAvatar> ghostAvatars;

  public ProtocolClient(InetAddress remAddr, int remPort,
    ProtocolType pType, MyGame game) throws IOException
  { super(remAddr, remPort, pType);
    this.game = game;
    this.id = UUID.randomUUID();
    this.ghostAvatars = new Vector<GhostAvatar>();
  }

  @Override
  protected void processPacket(Object o)
  { String message = (String) o;
    String[] msgTokens = message.split(",");

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

      if ((msgTokens[0].compareTo("dsfr") == 0) // receive "dsfr"
        || (msgTokens[0].compareTo("create") == 0))
      { // format: create, remoteld, x,y,z or dsfr, remoteld, x,y,z
        UUID ghostID = UUID.fromString(msgTokens[1]);
        Vector3 ghostPosition = Vector3f.createFrom(
          Float.parseFloat(msgTokens[2]),
          Float.parseFloat(msgTokens[3]),
          Float.parseFloat(msgTokens[4]));

        try
        { createGhostAvatar(ghostID, ghostPosition);
        } catch (IOException e)
        { System.out.println("error creating ghost avatar");
        }
      }

      if(msgTokens[0].compareTo("wsds") == 0) // rec. "create..."
      { // etc..... }
      if(msgTokens[0].compareTo("wsds") == 0) // rec. "wants..."
      { // etc..... }
      if(msgTokens[0].compareTo("move") == 0) // rec. "move..."
      { // etc..... }
    }
  }
}
```

Also need functions to instantiate ghost avatar, remove a ghost avatar, look up a ghost in the ghost table, update a ghost's position, and accessors as needed.

```
public void sendJoinMessage()    // format: join, localld
{ try
  { sendPacket(new String("join," + id.toString()));
  } catch (IOException e) { e.printStackTrace();
  } }
```

```
public void sendCreateMessage(Vector3 pos)
{ // format: (create, localld, 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();
  } }
```

```
public void sendByeMessage()
{ // etc..... }
public void sendDetailsForMessage(UUID remId, Vector3D pos)
{ // etc..... }
public void sendMoveMessage(Vector3D pos)
{ // etc..... }
```

public class GhostAvatar

```
{ private UUID id;
  private SceneNode node;
  private Entity entity;

  public GhostAvatar(UUID id, Vector3 position)
  { this.id = id;
  }

  // accessors and setters for id, node, entity, and position
  ...
}
```

Game:

```
...
import ray.networking.IGameConnection.ProtocolType;
// "sm" refers to the SceneManager
...
public class MyGame extends VariableFrameRateGame
{
    ...
    private String serverAddress;
    private int serverPort;
    private ProtocolType serverProtocol;
    private ProtocolClient protClient;
    private boolean isClientConnected;
    private Vector<UUID> gameObjectsToRemove;
    ...

    public MyGame(String serverAddr, int sPort, String protocol)
    {
        super();
        this.serverAddress = serverAddr;
        this.serverPort = sPort;
        if (protocol.toUpperCase().compareTo("UDP") == 0)
            this.serverProtocol = ProtocolType.UDP;
        ...
    }

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

    private void setupNetworking()
    {
        gameObjectsToRemove = new Vector<UUID>();
        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();
        }
    }

    protected void update(Engine engine)
    {
        // same as before, plus process any packets received from server
        ...
        processNetworking(elapsTime)
        ...
    }

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

        // remove ghost avatars for players who have left the game
        Iterator<UUID> it = gameObjectsToRemove.iterator();
        while(it.hasNext())
        {
            sm.destroySceneNode(it.next().toString());
        }
        gameObjectsToRemove.clear();
    }
}
```

```
public Vector3 getPlayerPosition()
{
    SceneNode dolphinN = sm.getSceneNode("dolphinNode");
    return dolphinN.getWorldPosition();
}

public void addGhostAvatarToGameWorld(GhostAvatar avatar)
    throws IOException
{
    if (avatar != null)
    {
        Entity ghostE = sm.createEntity("ghost", "whatever.obj");
        ghostE.setPrimitive(Primitive.TRIANGLES);
        SceneNode ghostN = sm.getRootSceneNode().
            createChildSceneNode(avatar.getID().toString());
        ghostN.attachObject(ghostE);
        ghostN.setLocalPosition(desired location...);
        avatar.setNode(ghostN);
        avatar.setEntity(ghostE);
        avatar.setPosition(node's position... maybe redundant);
    }
}

public void removeGhostAvatarFromGameWorld(GhostAvatar avatar)
{
    if(avatar != null) gameObjectsToRemove.add(avatar.getID());
}

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

Avatar movement (in input action class):

```
import ray.input.action.AbstractInputAction;
import ray.rage.scene.*;
import ray.rage.game.*;
import ray.rml.*;
import net.java.games.input.Event;

public class MoveForwardAction extends AbstractInputAction
{
    private Node avN;
    private ProtocolClient protClient;

    public MoveForwardAction(Node n, ProtocolClient p)
    {
        avN = n;
        protClient = p;
    }

    public void performAction(float time, Event e)
    {
        avN.moveForward(0.01f);
        protClient.sendMoveMessage(avN.getWorldPosition());
    }
}
```