SEECS

Distributed Computing

Assignment #01: Answers

Ammarah Tauheed – 2010-NUST-SEECS-BE-SE-281
Faria Kalim - 2011-NUST-SEECS-BE-SE-259
Maryam Tauheed - 2011-NUST-SEECS-BE-SE-272

Questions:

I. *Evaluate the portion of your design that deals with starting, maintaining, and exiting a game - what are its strengths and weaknesses?*

Starting the Game

Strengths:

- The beginning of the game is very simple – a new player looks at the packets that are multicast by other players, increments the global ID, assigns itself this new ID, and sends a packet with the new global ID to all other users.

Weaknesses:

- To assign an ID to the very first player, the game must timeout 25 times. This is a relatively long period of time - the assignment of the first ID should be faster, as until the first ID is assigned, nothing useful can happen in the game.

Maintaining the Game

Strengths:

- If a player loses contact with all other players, it is assumed that all other players have left the game (given that we wait for 29 timeouts to occur after no packet is received, this does not seem like an unreasonable assumption). Then, the player becomes the 0^{th} player, and the game starts over.
- The tagging player sends the tagged player a 'tagged' message to tell them that he has been tagged. Until the tagged player sends the tagging player an acknowledgement that they have received the message, this message is continuously sent out. Thus, we maintain consistency in the game. (All duplicates received by the tagged player are discarded.)
- We discard duplicates, and handle reordering of packets by using sequence numbers to ensure reliability. However, we do not concern ourselves with the corrupted packets or the loss of heartbeat. Only if a 'tagged' packet (described above) is lost, we resend it and stop resending when we receive an acknowledgement that it has been received.

Weaknesses:

- Let us consider a scenario where only player 6 and 7 are playing and all the others have quit. Another incoming player will not be able to play the game, as it will see that the next global ID available is 8, and that means that there are already 8 players playing the game (which is not true). This can be fixed

by employing a mechanism that changes the ID of the players still playing the game.

- If there is only one player in the game, then they would be reassigned ID 0 after every 29 timeouts when no packets are received. This may be unnecessary. This can be fixed by using a flag in the game to show that we have just started again, and need not reassign an ID.
- A lot of network traffic is incurred as we send out a packet once every iteration of the while(TRUE) loop in the game. For all 8 players, this would slow down the game. This can be improved by sending fewer status messages.

Exiting the Game

Strengths

- A player can leave a game very simply by sending out 'leaving' packets. For all other players, the process that checks if all players are participating shall automatically clean up this player's information from the record.
- If the 'leave' packet is lost, this does not make a difference to the way the game is played, as the program is written in such a way that if a player has not sent a packet in 29 timeouts, then they are automatically removed from the game.

Weaknesses

There are no significant weaknesses that we could identify.

General Strengths and Weaknesses

Strengths

- The game is programmatically simple – there is only one packet type that can be used for all purposes.

Weaknesses

- We could not get ClearRatPosition to work for other players, when a player has left the game
- We could not correctly use ConvertIncoming, because the packet received at the receiver's end (after having already passed through ConvertOutgoing at the sender's end) was already in the correct form.
- We assume that all packets received are in the correct format and are not corrupted. Therefore, if a corrupted packet was received, the game would probably throw an exception. This can be corrected by using some form of checksum.

- There are fields in the packet that are not regularly updated such as the rat name or tagged rat. These can be removed by making several types of packets instead of just one.

II. *Evaluate your design with respect to its performance on its current platform (i.e. a small LAN linked by ethernet). How does it scale for an increased number of players? What if it is played across a WAN? Or if played on a network with different capacities?*

Current Game:

A lot of packets are multicast in the game (one status packet per iteration of the while(TRUE) loop). This could slow down the game when all 8 players are playing the game.

However, since there are no packet losses, all status messages are promptly received and any delay in the game is avoided.

Scaling for More Players:

The game does not scale well for more than 8 players due to the network traffic that is generated.

However, in terms of memory usage, the game would scale well for any number of players, as we are only maintaining a few constructs for the players.

With respect to computational performance, the only time consuming operations are those which involve looping through all the players in the game. We can improve this by using threads for such computations, with the number of threads used depending on the number of players.

Playing Across a WAN/a Network with Different Capabilities:

The time required to transfer packets across a WAN or a heterogeneous network may bring a lot of delay in the game. Currently, the game works like this: if player A has tagged player B, player A is the one who will know first that B has been tagged. Then A will send B a packet letting them know they have been tagged. Over a WAN, B may receive the packet after a delay, during which B may have moved into another corridor from A. Thus, to B, it would seem that it was tagged even when there was no one else in the corridor, which does not make sense.

This can be fixed by adding the coordinates of the projectile to the packet, and every player can evaluate themselves whether they are hit or not in the game. Then they can exchange a message sequence that would ensure that both parties know who they tagged a player and got tagged respectively.

III. *Evaluate your design for consistency. What local or global inconsistencies can occur? How are they dealt with?*

Global Inconsistencies

- **Having the same ID**

Two players could be assigned the same ID if their games are instantiated at exactly the same time. This would happen because they would both see the same current global ID and would increment it to the same number, to assign an ID to themselves. Since the probability of this happening is very little, this has not been dealt with.

- **Standing at the same location**

Two players can stand at the same location in the game. This is dealt with separately in each game without sending packets, by making sure that if a player is in front of a player in the game, then the player trying to move forward should not be able to do so.

- **Tagging**

The 'tagged' packet is sent repeatedly to the tagged player until the tagged player acknowledges that they've been tagged. This ensures that the tagging information is known consistently across all games.

Local Inconsistencies

We could not identify any local inconsistencies.

IV. *Evaluate your design for security. What happens if there are malicious users?*

Since there are no security checks, malicious users could wreak havoc in the game.

The worst that they could do, in terms of the game, would be to claim that they have tagged other players and send other players tagged packets, hence winning very easily. They could also update the global ID to 7, preventing new players from joining the game.

In terms of the way the game is coded, malicious users could send corrupted packets or packets with incorrect formats, causing the games of other player's to throw exceptions and stop playing. They could also flood the network with status packets or use the IDs of other players to send messages.

This can be dealt with by applying checks for the packet format and using any error detection method to check for corrupted packets. Some intelligence could also be introduced in the game to detect malicious users and dropping their packets or removing them from the game altogether.