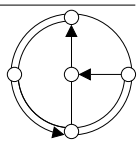
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich



WS 2005-2006

Roger Wattenhofer, Nicolas Burri, Pascal von Rickenbach,
Yves Weber, and (Andreas Wetzel)

Mobile Computing
Exercise 7

Assigned: January 9, 2006
Due: January 23, 2006

Hearts Lobby

So far we have implemented the multi hop communication layer for the hearts game we want to make ad hoc network compatible. Now it is time to use this communication layer to implement the game related components.

The first task we have to solve before we can start a Hearts game is to find four players who are interested in playing together. For this purpose, in this exercise we implement a lobby system where players can host and join games. To do so, we define a new communication layer (HeartsCommunication) on top of the multi hop layer.

1 The HeartsCommunication layer

The communication algorithm of the lobby is similar to Source Routing but with more human interaction: A client who is interested in finding an open game starts a flooding through the network requesting information about open games. Every client hosting a not yet started game responds to this flooding by sending back a message containing the necessary information about its open game. Then, the new player selects one of the found games and sends a join request to the host of this particular game. If the game is still missing at least one more player, the host replies with a join acknowledgment. Finally, once four players have joined a game the host can start the actual Hearts game by create a start message to all clients.

The message format used by the lobby is defined in the file `HeartsMessage.java` which can be downloaded on the course website. It basically consists of an `int` value containing the HeartsMessage type[1] and an `Object` which is used to add a custom parameter to the message. In the following it is described how the lobby needs to send and react upon the different HeartsMessage types.

Looking for open games

To look for open games a HeartsMessage of type `LOOK_FOR_GAMES` (0x01) has to be flooded through the network. Choose a reasonable TTL for the flooding![2] The `payload` field of the HeartsMessage can be set to `null`.

[1] Not to be confused with the type of the multi hop layer. All messages sent by the HeartsCommunication are of type TYPE_SRMSG on the multi hop layer. The whole HeartsMessage is put into the data field of the multi hop message.

[2] Flooding of a message of type TYPE_SRMSG was not explicitly specified in the last exercise. However, it should not be difficult to add this functionality to your solution. Simply check if a message of type TYPE_SRMSG has the broadcast address as a destination and deal with it as if it was a route search message. However, don't forget to hand the message to the handler for TYPE_SRMSG messages.

Reacting on a LOOK_FOR_GAMES message

On receiving a message of type LOOK_FOR_GAMES (0x01) the client checks if it is hosting a game. If it does host a game which does not have four joined players yet, it replies by sending a HeartsMessage of type HAVE_GAME (0x02) with the corresponding Game object as payload. The message is sent to the originator of the flooding which can be extracted from the sender field of the multi hop message. If the node does not host an open game it can ignore the message.

Joining a game

To join a previously found game a client sends a HeartsMessage addressed to the host of this game. This message is of type JOIN_GAME_REQ (0x03) and contains the Player object of the player interested in joining the game as payload. Note that the client interested in joining the game is not yet a member of this game after sending the message.

Reacting on a JOIN_GAME_REQ message

If the locally hosted game can use another player the node receiving this message adds the player to its game. All necessary information can be found in the payload of the message. It then replies by sending a HeartsMessage of type JOIN_GAME_ACK (0x04) to the originator of the JOIN_GAME_REQ with the Game object of the local game as payload. It also sends a message of type HAVE_GAME (0x02) to all other players who have previously joined the game. This messages also have the Game object of the local game as payload.

Leaving a remote game

If a node wants to leave a not yet started game it sends a HeartsMessage of type LEAVE_GAME (0x05) to the host of the game. As payload the message contains the Player object of the player leaving the game.

Reacting on a LEAVE_GAME message

On receiving a message of type LEAVE_GAME the node hosting the game removes the player sending the message from its game. It then sends a message of type HAVE_GAME to the other joined players with the new Game object as payload.

Aborting a locally hosted game

If a node wants to abort a not yet started, locally hosted game, it sends a message of type ABORT_GAME (0x06) to all joined players. The message has the last valid Game object of this game as payload.

Reacting on an ABORT_GAME message

The nodes receiving an ABORT_GAME message change their local state to indicate that they are no longer a member of any game. They do not send out any messages.

Starting a game

Once four players have joined a game, the host can start the actual Hearts game. It sends a message of type START_GAME to all joined players with the corresponding Game object as payload. It would also start the local instance of the Hearts game. However, since we do not want to implement this part yet, the node may simply print a message to the console to indicate that everything worked as planned.

Reacting on a START_GAME message

On receiving a START_GAME message a node starts a local instance of the Hearts game. Since we do not implement the actual game in this exercise the nodes may only print a success message to the console.

Reacting on a HAVE_GAME message

Messages of type **HAVE_GAME** are used to inform players about the state of a game. If the node receiving such a message is looking for games, the message indicates a new game which needs some more players. If the player has joined a game and receives a **HAVE_GAME** message containing information about this game, the host wants to inform it about changes of the game state (e.g. a player has joined/left the game).

2 User Interface

Since the lobby system requires some user interaction, we need to provide a suitable user interface. Depending on your Java skills you may either write a console based interface or a small graphical tool.

3 Good-Bye Andi

With the beginning of this year Andreas Wetzel who has been in charge of the practical exercises has left DCG and started a new career in industry. We would like to thank him for all his help and wish him good luck in his new job!

If you have questions about the multi hop layer exercises you may still contact him but for questions about the new exercises please come to one of the remaining assistants.

Thanks
Nicolas