Prof. Dr.-Ing. Ralf Steinmetz
Multimedia communications Lab

Dipl. Inf. Robert Konrad
Polona Caserman, M.Sc.

TECHNISCHE UNIVERSITÄT DARMSTADT

„Game Technology“
Winter Semester 2017/2018

Solution 11

General Information

- The exercises may be solved by teams of up to three people.
- The solutions have to be uploaded to the Git repositories assigned to the individual teams.
- **The submission date (for practical and theoretical tasks) is noted on top of each exercise sheet.**
- If you have questions about the exercises write a mail to game-technology@kom.tu-darmstadt.de or use the forum at https://www.fachschaft.informatik.tu-darmstadt.de/forum/viewforum.php?f=557

P11 Practical Task: Multiplayer (5 Points)

In this exercise, we implement a basic multiplayer game. The exercise is based on "Superball" – you control a sphere at the bottom and have to evade spheres coming at you from above (there is no collision implemented though).

The code you are provided with opens a UDP socket to communicate over. You control the behavior of the program via the defines at the beginning of Exercise.cpp:

`#define MASTER` - Determines if the client is the master or the slave

`SRC_PORT, DEST_PORT` – The port numbers used to send and receive

`DEST_IP1 to DEST_IP4` – The parts of the IPv4 address of the destination (defaults to 127.0.0.1/localhost)

The controls are the following

Master: The master is controlled via the keys left, right, up, down

Slave: The slave is controlled via the keys w, s, a, d

The master is in control of the NPC-sphere and sends position updates for it to the slave client.

The master's sphere is controlled via the Boolean values `left, right, up, down`.

The slave's sphere is controlled via the Boolean values `left2, right2, up2, down2`.

Your task is to send packets to the other client to update these values, so that the spheres carry out the same movements on both clients. You can use the function `sendPacket` for this.

Practical note: One approach for working on this exercise is to check out the files in two separate folders, one for the master and one for the slave. Be careful about the projects that `Kore/make` generates – they usually refer to the source files with absolute references, so just copying the folders after calling `Kore/make` will not create two separate projects. Best call Kore/make again in each folder. Please make sure that you keep the source code identical between the two versions.

https://github.com/TUDGameTechnology/Exercise11.git contains additional code to help you out. You can either copy the code changes manually or just pull them into your own repository using
git pull **https://github.com/TUDGameTechnology/Exercise11.git**

Beware, Exercise11 contains two korefiles – cd into Client and Server and each time execute *node ../Kore/make*

Client/build will then contain project files for the client and Server/build will contain project files for the client.

Please remember to push into a branch called exercise11.

You can find the solution code for the practical tasks at
https://github.com/TUDGameTechnology/Solution11.git.

T11 Theoretical Tasks: Multiplayer (5 Points)

T11.1 Peer-to-Peer drop in (2 points)

In the Peer-to-Peer Lockstep model clients can't drop in or out while the game runs. Describe a modification of the model that allows clients to join while the game runs.

Peer-to-peer-lockstep is based on the concept that each client simulates the game state and advances the state based on the input of all players. Players can't join a running game because they do not know the current game state.
To allow players to join, we would have to provide the current game state to the joining client. This can be done in two ways.

1) We keep a list of all the previous inputs that were received. Upon joining, the client receives this list and advances the game state to this state. When it is up to date with the game session, it can join in.

2) One of the already playing clients serializes the game state and sends it to the joining player.

In practice, also a mixture of both ways could be possible: The client receives a snapshot using method 2), which takes a certain amount of time during which the other clients continue playing. Then, the client receives the missing inputs to advance to the current game state.

T11.2 Analysis (2 points)

The source code provided for you in Task 1 does not implement pure Lockstep Peer-to-Peer Multiplayer. Name two aspects of the approach that differ from Lockstep Peer-to-Peer as presented in the lecture.

1) *The NPC ball is controlled by the Master and sent to the Slave. This is the client/server approach of multiplayer. For it to be a lockstep multiplayer approach, the two clients would each have to simulate a local version of the NPC ball.*
2) *There is no lockstep. The two clients send commands as fast as they can and do not wait for acknowledgements. Also, the movement speed is dependent upon the local framerate. Therefore, over time and using an internet connection with jitter and packet loss, the positions of the spheres between clients will start to diverge.*

Seen in total, the code is a mixture between client/server and peer-to-peer multiplayer. Sending key inputs to the other clients is a part of lockstep multiplayer, while the NPC sphere being controlled by the master is a part of the client/server approach.

T11.3 Varying data rates (1 Point)

Some network connections are fast and some are slow.

a) Can Peer-to-Peer Lockstep games handle varying network speeds? If so, how? **(0.5 Points)**

When the connection is too slow the complete game will slow down for all clients – Peer-to-Peer games cannot properly handle this situation because the network speed is directly linked to the execution of the game loop. Running the game loop less often when less data arrives would break most games (see for example tunneling in collision routines).

b) Can Client/Server games handle varying network speeds? If so, how? **(0.5 Points)**

Yes, this is implicitly supported. Clients always interpolate game states anyway; the game just runs nicer with smaller steps.