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

Dr. Florian Mehm
Dipl. Inf. Robert Konrad

TECHNISCHE UNIVERSITÄT DARMSTADT

„Game Technology“
Winter Semester 2016/2017

Exercise 12

For bonus points upload your solutions until **Saturday the 28th of January, 9:50**

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

P12 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/Exercise12.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/Exercise12.git`**

Please remember to push into a branch called exercise12.

T12 Theoretical Tasks: Multiplayer (5 Points)

T12.1 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.

T12.2 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.

T12.3 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)**

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