Distributed Systems Project-Report for Conflict:

Pascal Widmer	Julian Fuchs	Stefan Blumer
10-924-439	10-927-200	11-921-749
pawidmer@student.ethz.ch	fuchsju@student.ethz.ch	blumers@student.ethz.ch

1. Introduction:

For this assignment we used the Nexus One with Android 4.2 as the virtual android device. Since all three of us have the same phone, namely the Galaxy S II, we tested everything on them with Android version 4.1.2. We compiled using version 4.3.

Summary

For our final distributed systems project we developed a space shooter named "Conflict". Players can fight in space simultaneously and in real time. Every player owns a spaceship and can roam freely in a large square space. Every spaceship is equipped with a weapon and can fire projectiles on demand. Clicking on the left side of the screen causes the spaceship to accelerate while clicking on the right side fires projectiles. The spaceships direction can be changed by tilting the phone. Hitting another player with a projectile will cause him to be disconnected from the server.* The goal of our little game is to be the last man standing.

*TODO

2. Structural Overview

General Idea

The game consists of a map of variable size which no spaceship can leave. This map is further segmented into smaller tiles. Each of these tiles contains game objects, i.e. spaceships or projectiles. The game server keeps track of all objects on all tiles. The server handles clients joining the game and calculates all game interactions. The server is running on a pc.

Joining the game

Every android device connects to the server (currently a fixed IP). Upon receiving a message from a client, the server creates a spaceship and places it on the map.

Playing

The game server sends constant updates with positions of game objects to all connected clients, but each player only gets data which corresponds to objects in his proximity. The clients then display the received data on their Android devices and answer the update by sending the user input to the server. The server evaluates the input and runs its calculations. This process repeats itself as long as the game session lasts.

Project Structure

Our game consist of four projects named "Conflict", "Conflict-Android", "Conflict-desktop" and "Conflict-Server". The "Conflict-Server" gets started once on a PC, and then every player starts the "Conflict-Android" app on their android device and connects to the server. Both, "Conflict-Android" as well as "Conflict-Server" use the methods defined in our "Conflict"-Project. Thus, the "Conflict"-Projects acts like a library. Lastly, there is the class “Conflict-desktop”, which is still a work in progress, but should be a PC-port of our game.

3. Implementation Details

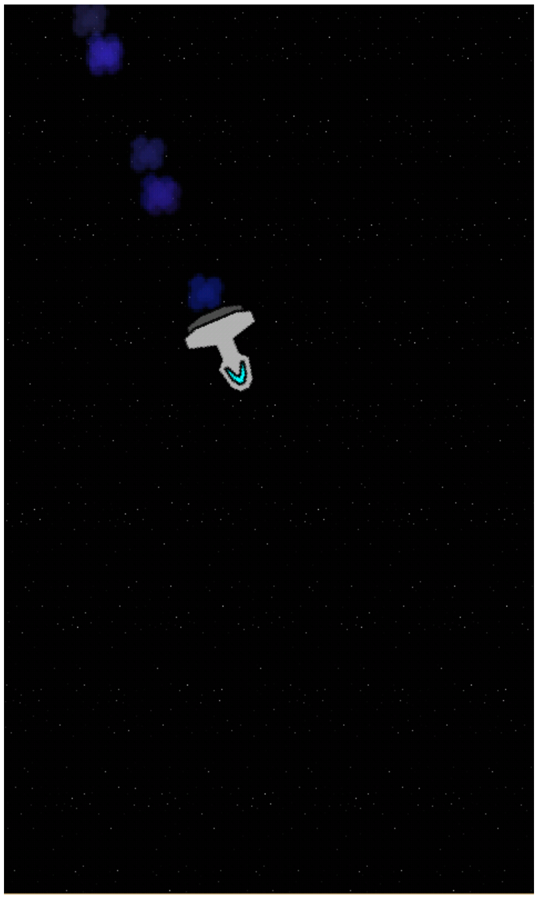
Graphics and Input

For generating graphics and handling player input we used the “libgdx” library. The spaceship can be accelerated by clicking on the left side of the screen and by clicking on the right side projectiles are launched. The spaceships direction is changed by tilting the phone. All game objects including the particles emitted from the spaceship are rendered using sprites. Each client receives 5 updates from the server every second and they include the spaceships position, positions of nearby spaceships and positions of projectiles. To make the animations more fluid we used a linear interpolation of 2 consecutive updates.

Networking

Our game uses UDP to communicate. At first we used TCP but it turned out to be too slow for real time applications such as our game. TCP uses "Nagle's algorithm" which improves the efficiency at the expense of latency.
It works by buffering small messages until a full packet can be sent or the last packet is acknowledged. UDP on the other hand sends small messages immediately but does not guarantee their arrival.
The game server keeps track of all connected clients and sends updates to each of them 5 times a second. Every client responds to every update. If the server does not get any reply from a client for a few seconds the client is declared dead and is removed from the list of active players. Losing a UDP packet every now and then leads to lag and possibly some action is lost, but because the server handles all player interactions the game will eventually be in a consistent state.

For sending objects we implemented our own serialization and deserialization codes. This was done to minimize the amount of data sent over the network since the default java serialization code generates too much overhead for our task.



4. Conclusion

Future improvements

The game could easily be expanded by adding different types of weapons, ships or landscapes. A projectile impact could for instance be visualized with an explosion animation. A simple GUI and some sound effects would also make the game more fun and user-friendly. We did not focus on these things because they are simple, yet very time consuming to do and also because graphics is not a central part of this course.