Program Architecture

This document gives an outline of the overall program architecture. This includes package structure, networking and game flow. In each section, a detailed description of the processes, classes and relations is given.

Packages

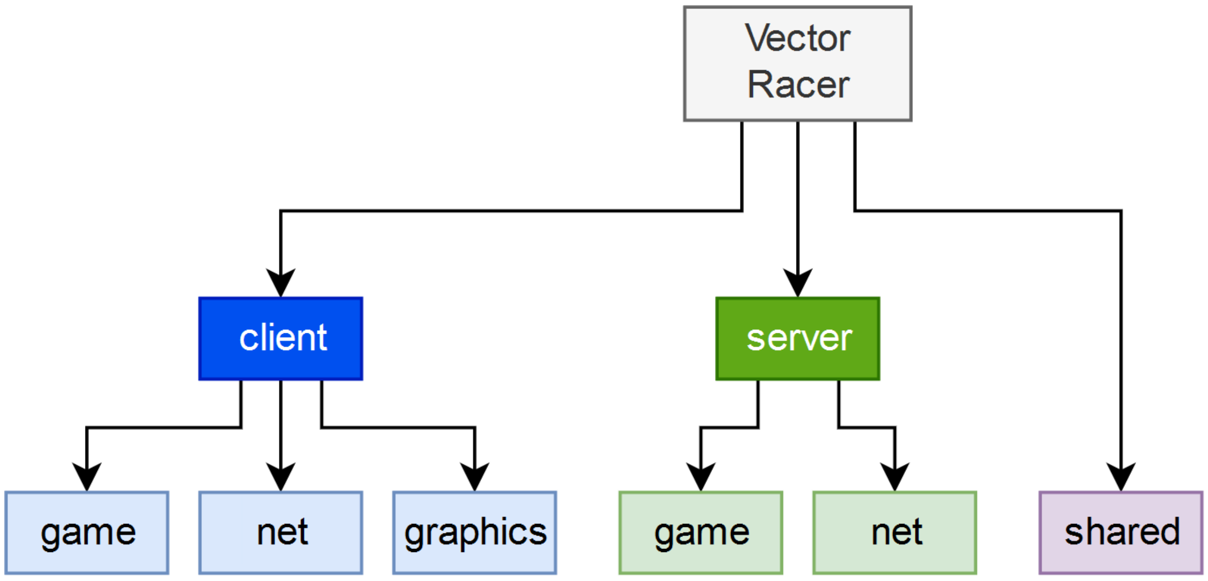

Figure 1: Package Structure

The package structure of our project is visualized by the graph above. The main module *Vector Racer* is divided into 3 sub-packages: *client*, *server* and *shared*. Classes are only stored in the lightly shaded packages.

The game package of both *client* and *server* are responsible for handling information of games such as progress, players, standings and rules. The *client* package is responsible for local processing and the *server* package receives, processes and distributes game information to the according players.

'net' stands for networking in the sub-packages of *client* and *server*. The classes inside of the packages handle all communication tasks between a client and a given server. This includes sending, processing and receiving moves, messages, events and commands.

All classes within the *client.graphics* package are required to render everything associated with Game UI, menu and chat.

The shared package contains classes accessed by both, client and server, i.e. the game launcher.

Networking

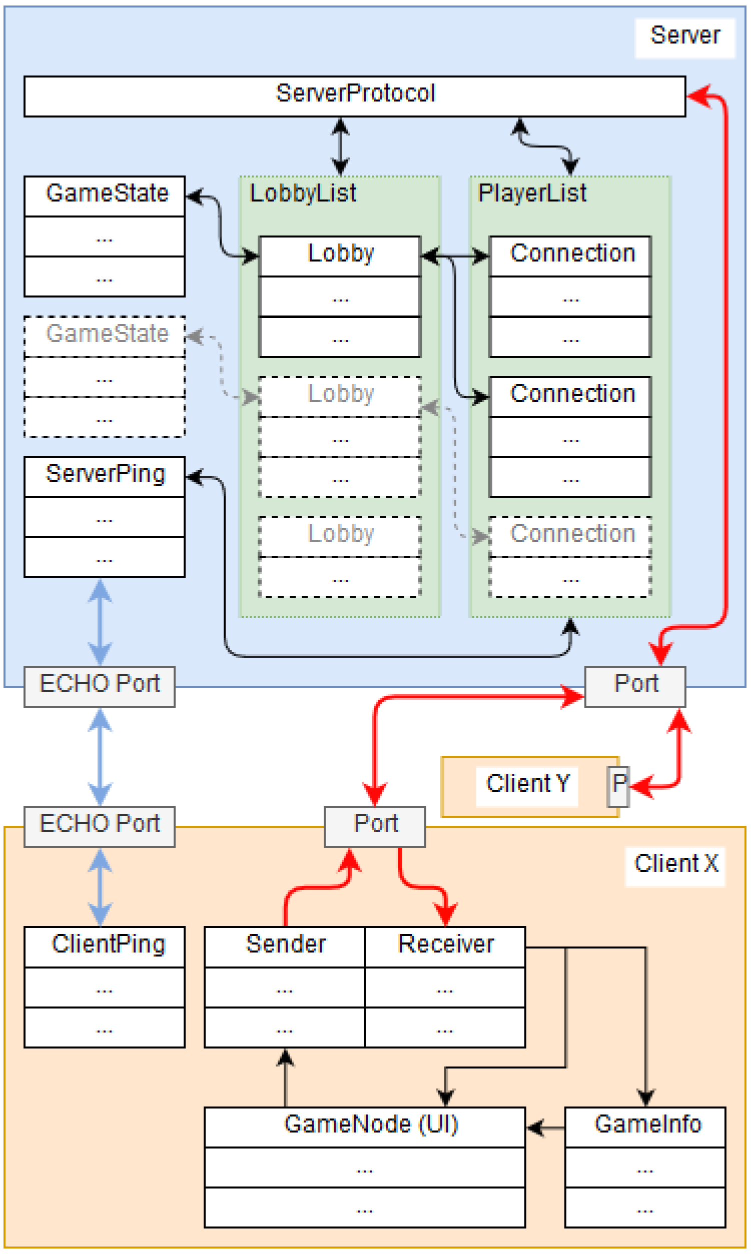

Figure 2: Network Overview

To the left, a simplified overview of the networking can be seen.

The red arrows represent the main communication channel between server and clients. As shown, each client has a sender and a receiver class whereas the server unifies the asks into one class, ServerProtocol.

The black arrows represent local communication between classes and processes.

The server is listening for connections on a given port and upon establishing a connection, a new client-handling thread is created and added onto the player list.

Connected players can create lobbies, which are stored in a list as well. Players can bind/unbind themselves to a lobby by joining/leaving said lobby.

For each lobby, a GameState object is created and instantiated upon start of a lobby-internal game.

The player is not able to directly access the sender and receiver, because all interaction is handled by the UI class, GameNode.

Using the UI, the player can utilize all chat, lobby, game and general functions.

The classes Receiver and ServerProtocol decode and process incoming information and depending on the content, takes the corresponding action(s).

The blue arrow indicates communication required for the ping classes. For both, server and client, the port 7, which corresponds to the ECHO port, is pinged to ensure the availability of a device.

Game flow

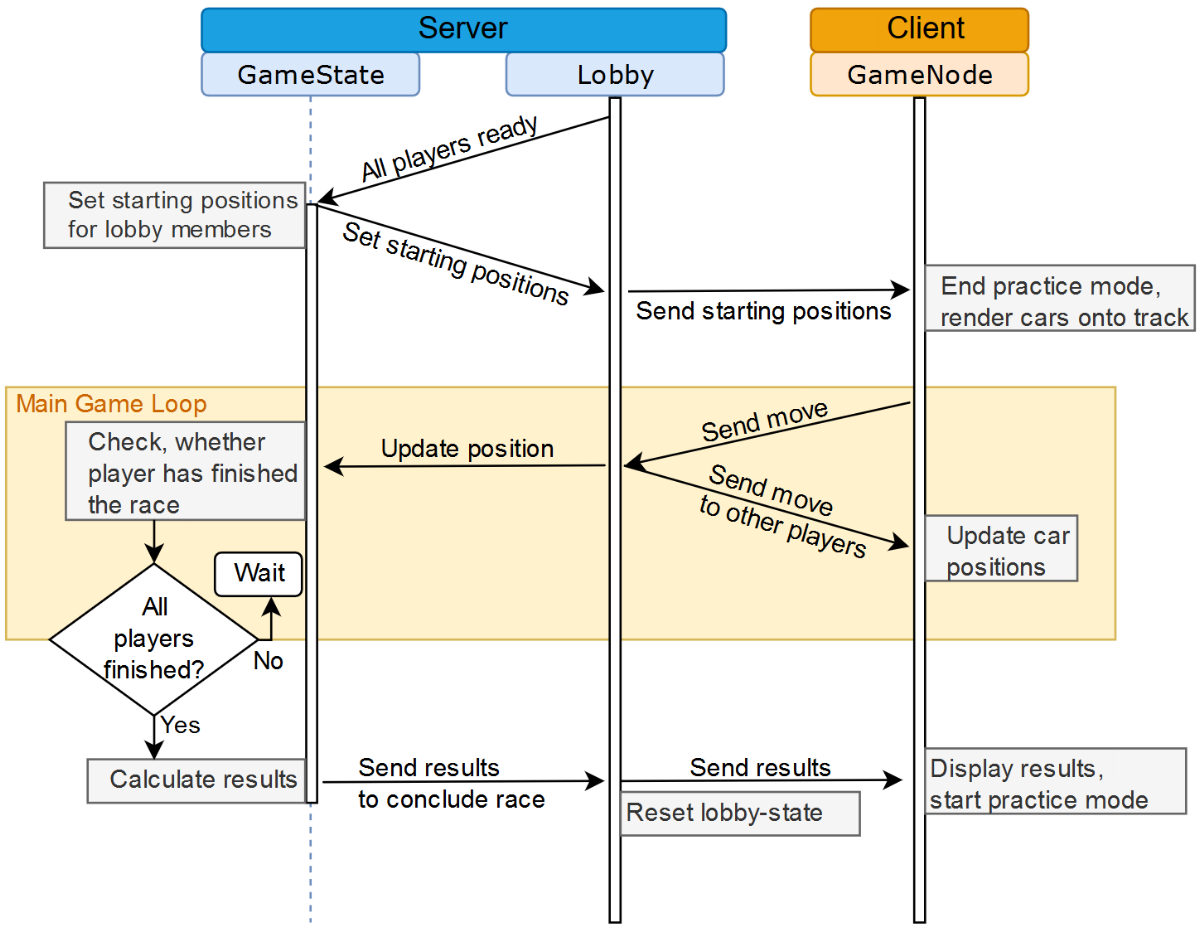

Figure 3: Game Flow

The graph above describes the processes during an active race. Every client can toggle their ready status by using the designated button. When all players are ready, the lobby class instantiates a GameState object.

When the last player has crossed the finish line, the results will be displayed onto the chat of all lobby members.

Note that the game flow of the project also ensures, that whenever a player is not in an active race, the practice mode is activated.